National University of Singapore

CS3236

Introduction to Information Theory

Lecture Notes

April 2026

Contents

Lecture 1: Introduction

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
- 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

Assessment weightage:

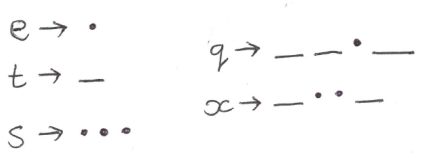
- Final: 40%
- Midterm: 20%
- Weekly assignments: 20%
- Tutorial participation: 10%
- Class quizzes: 10%

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 it 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

  $$0000000\mathbf{1}000000000000000000000\mathbf{1}000000000000000000000000\mathbf{1}000000000,$$

  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

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

  (More generally, compressing N different strings down to L bits without loss requires $L \geq \log_2 N$.)

- 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 $P_{X_1 X_2 \ldots X_{999} X_{1000}}$, where each X_i takes some value in $a \ldots z$ (or space).
  - Probabilistic modeling can provide a very good trade-off between accuracy in modeling the real world vs. tractability of the mathematical analysis.
- 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.

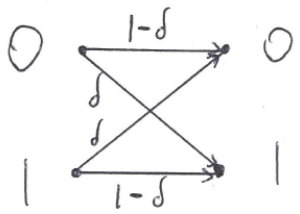
3 A Preview of Data Communication

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 000000011111110000000111111 (repetition code R_7)
  - Any given bit (out of the 4 sent) is decoded correctly with probability

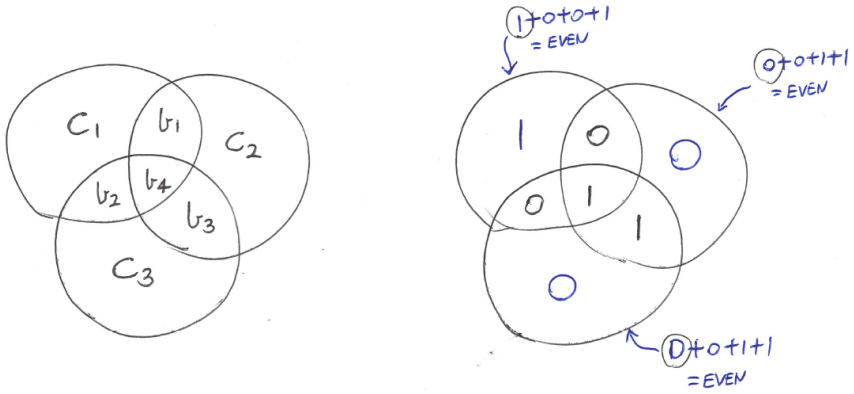
$$(1-\delta)^7 + 7\delta(1-\delta)^6 + \binom{7}{2}\delta^2(1-\delta)^5 + \binom{7}{3}\delta^3(1-\delta)^4 \approx 0.9973$$

    (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

Lecture 2: Useful Probability Properties

- **General notation:**
  - If X if a discrete random variable, its probability mass function (PMF) is written as P_X (i.e., $P_X(x) = \mathbb{P}[X = x]$).
  - If X is a continuous random variable, its probability density function (PDF) is written as f_X (e.g., $\mathbb{P}[a \leq X \leq b] = \int_a^b f_X(x)dx$). **For now (and most of the course), let's assume all random variables are discrete.**
  - We will usually use uppercase for a random variable (e.g., X) and the corresponding lower-case letter for a specific value (e.g., x).
- **Expectation:**
  - Definition: $\mathbb{E}[X] = \sum_x P_X(x)x$
  - Average of function: $\mathbb{E}[f(X)] = \sum_x P_X(x)f(x)$ for deterministic f
  - Average of scaled RV: $\mathbb{E}[cX] = c\mathbb{E}[X]$ for constant c
  - Average of sum: $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$ regardless of whether or not X and Y are independent
  - Average of product: If X and Y are independent, then $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$
  - Indicator function: If $\mathbf{1}\{A\}$ denotes the indicator function (equaling 1 if the event A holds and 0 otherwise), then $\mathbb{E}[\mathbf{1}\{A\}] = \mathbb{P}[A]$
- **Conditioning:**
  - Definition: $P_{Y|X}(y|x) = \frac{P_{XY}(x,y)}{P_X(x)}$
  - Law of total probability: For an event A and RV X, we have $\mathbb{P}[A] = \sum_x P_X(x)\mathbb{P}[A|X = x]$
  - Law of total expectation: (AKA tower property) $\mathbb{E}[Y] = \mathbb{E}\big[\mathbb{E}[Y|X]\big]$, where the outer expectation is over X and the inner one is over Y (given X)
  - Bayes' rule: $\mathbb{P}[A|B] = \frac{\mathbb{P}[A]\mathbb{P}[B|A]}{\mathbb{P}[B]}$

- **Independence:**
  - Definition: $P_{XY}(x,y) = P_X(x)P_Y(y)$ for all x, y
  - Equivalent definition 1: $P_{Y|X}(y|x) = P_Y(y)$ for all x, y
  - Equivalent definition 2: $P_{X|Y}(x|y) = P_X(x)$ for all x, y
  - Analogous definitions for conditional independence: (i) $P_{XY|Z}(x,y|z) = P_{X|Z}(x|z)P_{Y|Z}(y|z)$ for all x, y, z; (ii) $P_{Y|XZ}(y|x,z) = P_{Y|Z}(y|z)$ for all x, y, z; (iii) $P_{X|YZ}(x|y,z) = P_{X|Z}(x|z)$ for all x, y, z. We use the terminology "X and Y are conditionally independent given Z".
  - Functions: If X and Y are independent, then so are $f(X)$ and $g(Y)$ for deterministic f, g
  - Conditional vs. unconditional: The statements "X and Y are independent" and "X and Y are conditionally independent given Z" can be very different:
    - Example 1: If X and Y are independent and $Z = X + Y$, then X and Y are certainly not conditionally independent given Z
    - Example 2: If U and V are independent and $X = Z + U$, $Y = Z + V$, then X and Y are conditionally independent given Z, but dependent due to the common reliance on Z
  - Joint independence of a collection $X_1, \ldots, X_n$ of random variables can be defined as $P_{X_1,\ldots,X_n}(x_1, \ldots, x_n) = \prod_{i=1}^n P_{X_i}(x_i)$.
    - Note: Pairwise independence does not necessarily imply joint independence
- **Variance:**
  - Definition: $\mathrm{Var}[X] = \mathbb{E}\big[(X-\mu)^2\big]$, where μ is the mean of X
  - Equivalent definition: $\mathrm{Var}[X] = \mathbb{E}[X^2] - \mu^2$
  - Scaling: $\mathrm{Var}[cX] = c^2\mathrm{Var}[X]$ for constant c
  - Variance of sum: If X and Y are independent, then $\mathrm{Var}[X+Y] = \mathrm{Var}[X]+\mathrm{Var}[Y]$. (More generally, $\mathrm{Var}[X+Y] = \mathrm{Var}[X] + \mathrm{Var}[Y] + 2\mathrm{Cov}[X,Y]$.)
  - Covariance: $\mathrm{Cov}[X,Y] = \mathbb{E}[(X-\mu_X)(Y-\mu_Y)]$ where $\mu_X = \mathbb{E}[X]$ and $\mu_Y = \mathbb{E}[Y]$
  - Law of total variance: (Not needed, see Wikipedia if interested)
- **Other:**
  - Marginal distribution: $P_X(x) = \sum_y P_{XY}(x,y)$ and similarly $P_Y(y) = \sum_x P_{XY}(x,y)$
  - Union vs. intersection: $\mathbb{P}[A \cup B] = \mathbb{P}[A] + \mathbb{P}[B] - \mathbb{P}[A \cap B]$
  - Union bound: (AKA Boole's inequality) $\mathbb{P}\big[\bigcup_{i=1}^N A_i\big] \le \sum_{i=1}^N \mathbb{P}[A_i]$
  - Law of large numbers: If $X_1, \ldots, X_n$ are independent and identically distributed (i.i.d.) with mean μ, then $\mathbb{P}\big[\big|\frac{1}{n}\sum_{i=1}^n X_i - \mu\big| > \epsilon\big] \to 0$ as $n \to \infty$ for arbitrarily small $\epsilon > 0$

- **Properties of logarithms (not related to probability):**
  - $\log xy = \log x + \log y$
  - $\log \frac{1}{x} = -\log x$
  - $\log \frac{y}{x} = \log y - \log x$
  - $\log x^c = c \log x$
  - $\log_a x = \frac{\log_b x}{\log_b a}$
  - $\log_e x \leq x - 1$ with equality if and only if $x = 1$
- **Very basic calculus (not related to probability):**
  - $\frac{d}{dx} x^c = cx^{c-1}$
  - $\frac{d}{dx} e^{cx} = ce^{cx}$
  - $\frac{d}{dx} \ln x = \frac{1}{x}$
  - Chain rule: $\frac{d}{dx} f(g(x)) = f'(g(x))\, g'(x)$
  - Product rule: $\frac{d}{dx}(f(x)g(x)) = f'(x)g(x) + f(x)g'(x)$
  - Quotient rule: $\frac{d}{dx}(f(x)/g(x)) = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$

Lecture 3: Information Measures

Useful references:

- Cover/Thomas Chapter 2
- MacKay Chapters 2–4

4 Information of an Event

Problem.

- If we are told that random event A occurred (e.g., a coin came up tails, two dice added up to 7, it rained today), how much "information" have we learned?
- Approach: Quantify information without any regard to *significance* or *importance*. It is only $\Pr[A]$ that matters.
  - Things like "importance" are usually too subjective to quantify.
- Generically speaking, if A occurs with probability p, then

$$\text{Information}(A) = \psi(p)$$

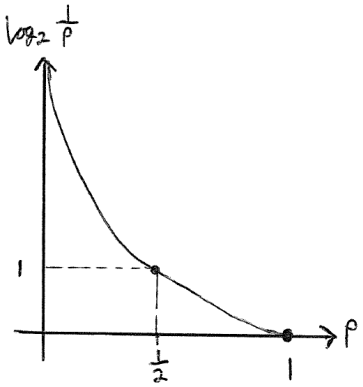
  for some function $\psi(\cdot)$. Perhaps a more intuitive interpretation of $\psi(p)$ is that it quantifies *how surprised we are that event A occurred*. What properties should this function satisfy?

Axiomatic view.

- Here are some very natural properties that we should expect $\psi(p)$ to satisfy:
  1. (Non-negativity) $\psi(p) \geq 0$, i.e., we cannot learn a "negative amount" of information.
  2. (Zero for definite events) $\psi(1) = 0$, i.e., if something was certain to happen, nothing is learned by the fact that it occurred.

3. (Monotonicity) If $p \leq p'$, then $\psi(p) \geq \psi(p')$, i.e., the less likely the event was, the more information is learned by the fact that it occurred.
4. (Continuity) $\psi(p)$ is continuous in p, i.e., small changes in probability don't cause drastic changes in information.
5. (Additivity under independence) $\psi(p_1 p_2) = \psi(p_1) + \psi(p_2)$. If A and B are independent events with probabilities p_1 and p_2, then $A \cap B$ has probability $p_1 p_2$, and the information learned from both A and B occurring is the sum of the two individual amounts of information (because they are independent!)

- It can be shown that only $\psi(p) = \log_b \frac{1}{p}$ (for some base $b > 0$) satisfies all five.
  - We focus on $b = 2$, which means information is measured in "bits". Another common choice is $b = e$, which means information is measured in "nats".
  - All choices of b are equivalent up to scaling by a universal constant (e.g., number of nats = $(\log_e 2) \times$ number of bits). This is much the same as how we can measure distance in meters, kilometers, inches, or miles, but converting from one to another just amounts to scaling.
  - So being told that a probability-p event occurred gives us $\log_2 \frac{1}{p}$ "bits" of information.
  - An illustration:



5 Information of a Random Variable – Entropy

Definition.

- Let X be a discrete random variable with probability mass function (PMF) P_X
- According to the previous section, if we observe $X = x$ then we have learned $\log_2 \frac{1}{P_X(x)}$ bits of information. The **(Shannon) entropy** is simply the average of this value with

respect to P_X:

$$H(X) = \mathbb{E}_{X \sim P_X}\left[\log_2 \frac{1}{P_X(X)}\right] = \sum_x P_X(x) \log_2 \frac{1}{P_X(x)}.$$

- Note the convention $0 \log \frac{1}{0} = 0$, which is reasonable since $\lim_{p \to 0} p \log_2 \frac{1}{p} = 0$.

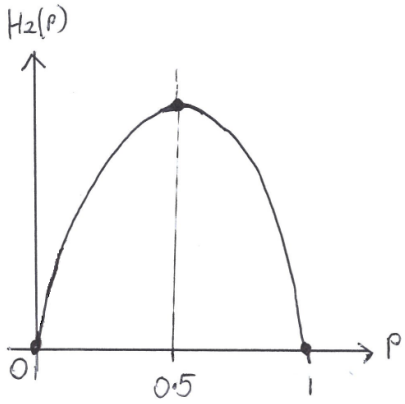
- Can be viewed as a measure of *information in X* or *uncertainty in X* (these are not contradictory)

- **Note.** Here and throughout the vast majority of the course, we only consider *discrete-valued* random variables that can only take on a finite number of values.

Examples.

- Binary source:

  - Suppose $X \sim \text{Bernoulli}(p)$ for some $p \in (0,1)$ (i.e., $P_X(1) = 1 - P_X(0) = p$)
  - Then we get
  $$H(X) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p}. \tag{1}$$
  The right-hand side, as a function of p, is known as the *binary entropy function*. Since this quantity will be used frequently throughout the course, we give it a formal definition: $H_2(p) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p}$ for $p \in [0,1]$. An illustration:



- Uniform source:

  - Suppose X is uniform on a finite set $\mathcal{X}$ (i.e., $P_X(x) = \frac{1}{|\mathcal{X}|}$ for each $x \in \mathcal{X}$, where $|\mathcal{X}|$ is the cardinality of $\mathcal{X}$)

- Then we get

$$H(X) = \mathbb{E}\left[\log_2 \frac{1}{1/|\mathcal{X}|}\right] = \log_2 |\mathcal{X}|.$$

This is intuitive, e.g., with 10 bits we can produce $|\mathcal{X}| = 2^{10}$ combinations of bits.

Axiomatic view [Shannon].

- Suppose that X is a discrete random variable taking N values, with probabilities $\mathbf{p} = (p_1, \ldots, p_N)$. If we consider a general information measure of the form

$$\Psi(\mathbf{p}) = \Psi(p_1, \ldots, p_N),$$

then what properties should it satisfy?

- Three natural properties:

1. (Continuity) $\Psi(\mathbf{p})$ is continuous as a function of $\mathbf{p}$. Again, small changes in the distribution don't give large changes in information/uncertainty.
2. (Uniform case) If $p_i = \frac{1}{N}$ for $i = 1, \ldots, N$, then $\Psi(\mathbf{p})$ is increasing in N. That is, being uniform over a larger set of outcomes always means more information/uncertainty.
3. (Successive decisions) The following always holds:

$$\Psi(p_1, \ldots, p_N) = \Psi(p_1 + p_2, p_3, \ldots, p_N) + (p_1 + p_2)\Psi\left(\frac{p_1}{p_1 + p_2}, \frac{p_2}{p_1 + p_2}\right).$$

This can be viewed as drawing from the distribution on X by first drawing from the corresponding distribution that doesn't distinguish two symbols (the ones with probabilities p_1 and p_2), and then drawing another random variable to resolve those two symbols if needed (which only happens a fraction $p_1 + p_2$ of the time). The total information/uncertainty is the sum of the information/uncertainty from each of the two stages.

- It can be shown that only $\Psi(X) = \text{constant} \times H(X)$ satisfies all three.

Variations.

- Joint entropy of two random variables (X, Y):

$$\begin{aligned} H(X, Y) &= \mathbb{E}_{(X,Y)\sim P_{XY}}\left[\log_2 \frac{1}{P_{XY}(X, Y)}\right] \\ &= \sum_{x,y} P_{XY}(x, y) \log_2 \frac{1}{P_{XY}(x, y)}. \end{aligned}$$

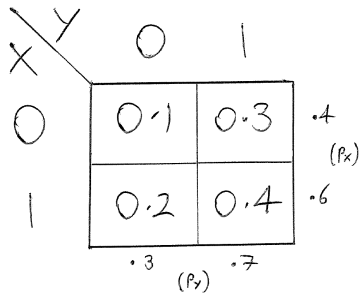
We can similarly define $H(X, Y, Z)$ or larger collections such as $H(X_1, \ldots, X_n)$.

- Conditional entropy of Y given X:

$$\begin{aligned} H(Y|X) &= \mathbb{E}_{(X,Y)\sim P_{XY}}\left[\log_2 \frac{1}{P_{Y|X}(Y|X)}\right] \\ &= \sum_{x,y} P_{XY}(x,y) \log_2 \frac{1}{P_{Y|X}(y|x)} \\ &= \sum_{x} P_X(x) H(Y|X=x), \end{aligned} \tag{2}$$

  where in the last line, $H(Y|X=x) = \sum_y P_{Y|X}(y|x) \log_2 \frac{1}{P_{Y|X}(y|x)}$ is simply the entropy of the distribution $P_{Y|X}(\cdot|x)$ on Y. We can similarly define quantities like $H(Y_1, Y_2|X_1, X_2)$.

  - Intuition: $H(Y|X=x)$ is the uncertainty in Y after having observed that $X = x$. The conditional entropy $H(Y|X)$ simply averages such a quantity over X, so it represents the average remaining uncertainty in Y after observing X.
  - Example: Consider the joint distribution described as follows.



    Each entry in the table is a $P_{XY}(x,y)$ value, and the values at the right and bottom are the resulting marginals $P_X(x)$ and $P_Y(y)$ (just add up the relevant row or column).

  - Combining the joint and marginal distributions gives the conditionals:

$$\begin{aligned} P_{Y|X}(0|0) &= \frac{P_{XY}(0,0)}{P_X(0)} = \frac{0.1}{0.4} = \frac{1}{4} \\ P_{Y|X}(0|1) &= \frac{P_{XY}(1,0)}{P_X(1)} = \frac{0.2}{0.6} = \frac{1}{3}. \end{aligned}$$

    Substitution into the form of $H(Y|X)$ in Eq. (2) gives

$$H(Y|X) = 0.4\, H_2\Big(\frac{1}{4}\Big) + 0.6\, H_2\Big(\frac{1}{3}\Big) \approx 0.8755,$$

    where $H_2(p)$ denotes the binary entropy function given on the right-hand side of Eq. (1).

    * Note that $H(Y|X)$ is smaller than $H(Y) = H_2(0.3) \approx 0.8813$ (on average, knowing X reduces uncertainty about Y)

* But $H(Y|X=1) = H_2\left(\frac{1}{3}\right) \approx 0.9183$ (seeing a *specific* outcome of X may increase uncertainty about Y)

5.1 Properties of Entropy

- **Non-negativity:**

$$H(X) \geq 0$$

with equality if and only if X is deterministic.

  - Intuition: Information/uncertainty cannot be negative
  - Proof: The "information of an event" $\log_2 \frac{1}{p}$ is always non-negative for $p \in [0,1]$, so entropy is the average of a quantity that is always non-negative, and so is itself non-negative. Moreover, only $p = 1$ gives $\log_2 \frac{1}{p} = 0$, so $H(X) = 0$ if and only if X is deterministic.

- **Upper bound:** If X takes values on a finite alphabet $\mathcal{X}$, then

$$H(X) \leq \log_2 |\mathcal{X}|$$

with equality if and only if X is uniform on $\mathcal{X}$. This similarly implies $H(X|Y) \leq \log_2 |\mathcal{X}|$.

  - Intuition: The uniform distribution has the most uncertainty.
  - Proof: Let P be the distribution of X, and let Q be the uniform distribution on $\mathcal{X}$, so that $Q(x) = \frac{1}{|\mathcal{X}|}$ for all x. Then note that

$$\begin{aligned}\sum_x P(x) \log_2 \frac{P(x)}{Q(x)} &= \sum_x P(x) \log_2 \left(|\mathcal{X}| \cdot P(x)\right) \\ &= \log_2 |\mathcal{X}| + \sum_x P(x) \log P(x) \\ &= \log_2 |\mathcal{X}| - H(X).\end{aligned}$$

    In Section 6 we will show that the left-hand side is non-negative for *any* distributions P and Q, with equality if and only if $P = Q$. Specialized to the above choices of P and Q, we get $\log_2 |\mathcal{X}| - H(X) \geq 0$ with equality if and only if P is uniform, as desired.

- **Chain rule (two variables):**

$$H(X,Y) = H(X) + H(Y|X)$$

  - Intuition: The overall information in (X,Y) is the information in X plus the remaining information in Y after observing X.

- Proof: For $(X, Y) \sim P_{XY}$, we have

$$\begin{aligned} H(X,Y) &= \mathbb{E}\left[\log \frac{1}{P_{XY}(X,Y)}\right] \\ &= \mathbb{E}\left[\log \frac{1}{P_X(X)P_{Y|X}(Y|X)}\right] \\ &= \mathbb{E}\left[\log \frac{1}{P_X(X)} + \log \frac{1}{P_{Y|X}(Y|X)}\right] \\ &= H(X) + H(Y|X). \end{aligned}$$

  - Note: We can swap the roles of X and Y, giving $H(X,Y) = H(Y) + H(X|Y)$.

- Chain rule (general):

$$H(X_1, \ldots, X_n) = \sum_{i=1}^{n} H(X_i|X_1, \ldots, X_{i-1}).$$

  - Intuition: Similar to the two-variable case.
  - Proof: Similar to the two-variable case, but instead use the expansion $P_{X_1 \ldots X_n} = P_{X_1} \times P_{X_2|X_1} \times P_{X_3|X_1X_2} \times \ldots \times P_{X_n|X_1,\ldots,X_{n-1}}$.

- Conditioning reduces[4] entropy:

$$H(X|Y) \leq H(X)$$

  with equality if and only if X and Y are independent.

  - Intuition: Having additional information cannot increase uncertainty *on average*.[5]
  - Proof: Equivalent to the property $I(X;Y) \geq 0$ to be proved in Section 7.1.

- Sub-additivity:

$$H(X_1, \ldots, X_n) \leq \sum_{i=1}^{n} H(X_i)$$

  with equality if and only if $X_1, \ldots, X_n$ are independent.

  - Intuition: The uncertainty in several random variables is no more than the sum of individual uncertainty in each one.
  - Proof: Apply "conditioning reduces entropy" to each summand in the general chain rule formula above.

[4] More precisely, does not increase

[5] In contrast, $H(X|Y = y)$ for a particular y could exceed $H(X)$, as we saw in the example following the conditional entropy definition (note that the roles of X and Y were reversed there).

6 A Useful Measure Between Distributions – KL Divergence

- For two PMFs P and Q on a finite alphabet $\mathcal{X}$, the *Kullback-Leibler (KL) divergence* (also known as *relative entropy*) is given by

$$\begin{aligned} D(P\|Q) &= \sum_x P(x) \log_2 \frac{P(x)}{Q(x)} \\ &= \mathbb{E}_{X\sim P}\left[\log_2 \frac{P(X)}{Q(X)}\right]. \end{aligned}$$

- Can be viewed as a kind of "distance" between P and Q, but it is not a distance function in the mathematical sense (in general it is not symmetric and doesn't satisfy the triangle inequality).

- **Claim.** For any distributions P and Q, we have

$$D(P\|Q) \geq 0$$

with equality if and only if $P = Q$.

  - Proof:

$$\begin{aligned} -D(P\|Q) &= \sum_x P(x) \log \frac{Q(x)}{P(x)} \\ &\overset{(a)}{\leq} \sum_x P(x)\left(\frac{Q(x)}{P(x)} - 1\right) \\ &= \sum_x Q(x) - \sum_x P(x) \\ &= 0, \end{aligned}$$

    where (a) uses the inequality $\log \alpha \leq \alpha - 1$, which is easily verified graphically. Equality holds in $\log \alpha \leq \alpha - 1$ if and only if $\alpha = 1$, which means that equality holds in (a) if and only if $\frac{Q(x)}{P(x)} = 1$ for all x (i.e., $P = Q$).

- The KL divergence (and in fact, also entropy and mutual information) is used extensively in other fields like statistics and machine learning. Some example uses (stated only very roughly here) are:

  - In data compression, if the true source is distribution is P but we use an algorithm that wrongly assumes it is Q, then we pay a penalty of $D(P\|Q)$ in the average number of bits per symbol;
  - In statistics, if $\mathbf{X} = (X_1, \ldots, X_n)$ is i.i.d. with $X_i \sim Q$, then the probability that $\mathbf{X}$ "looks like" it was generated i.i.d. on P (yes, this is quite vague) is roughly $2^{-nD(P\|Q)}$ when n is large. Look up *Sanov's theorem* for a more precise statement.

7 Information Between Random Variables – Mutual Information

Definition.

- Mutual information:

$$I(X;Y) = H(Y) - H(Y|X).$$

- Intuition:
  - $H(Y)$ is the *a priori uncertainty* in Y
  - $H(Y|X)$ is the *remaining uncertainty* in Y after observing X (on average)
  - Hence, $I(X;Y)$ is the *amount of information about Y we learn by observing X* (on average).

Variations.

- Joint version:

$$I(X_1, X_2; Y_1, Y_2) = H(Y_1, Y_2) - H(Y_1, Y_2|X_1, X_2).$$

- Conditional version:

$$I(X;Y|Z) = H(Y|Z) - H(Y|X, Z).$$

Examples.

1. If X and Y are independent, then it is straightforward to compute $H(Y|X) = H(Y)$, giving $I(X;Y) = 0$ (i.e., independent random variables do not reveal any information about each other).

2. If $Y = X$, then it is straightforward to compute $H(Y|X) = H(X|X) = 0$, and hence $I(X;X) = H(X)$ (i.e., the amount of information a random variable reveals about itself is the entropy).

3. In the example given shortly after Eq. (2), we computed $H(Y|X) \approx 0.8755$ and $H(Y) \approx 0.8813$, which gives $I(X;Y) = H(Y) - H(Y|X) \approx 0.006$.

4. We will see more "insightful" examples when we come to the channel coding (communication) part of the course.

7.1 Properties of Mutual Information

- **Alternative forms:**

$$\begin{aligned} I(X;Y) &= D(P_{XY} \| P_X \times P_Y) \\ &= \mathbb{E}\left[\log_2 \frac{P_{XY}(X,Y)}{P_X(X)P_Y(Y)} \right] = \sum_{x,y} P_{XY}(x,y) \log_2 \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)} \\ &= \mathbb{E}\left[\log_2 \frac{P_{Y|X}(Y|X)}{P_Y(Y)} \right] = \sum_{x,y} P_{XY}(x,y) \log_2 \frac{P_{Y|X}(y|x)}{P_Y(y)}. \end{aligned}$$

  - Proof: Substituting $H(Y) = \mathbb{E}\big[\log_2 \frac{1}{P_Y(Y)}\big]$ and $H(Y|X) = \mathbb{E}\big[\log_2 \frac{1}{P_{Y|X}(Y|X)}\big]$ into the definition of mutual information gives $I(X;Y) = \mathbb{E}\big[\log_2 \frac{P_{Y|X}(Y|X)}{P_Y(Y)}\big]$. Multiplying the numerator & denominator by $P_X(X)$ gives $\mathbb{E}\big[\log_2 \frac{P_{XY}(X,Y)}{P_X(X)P_Y(Y)}\big]$, from which the remaining equalities follow easily.

- **Symmetry:** We have

$$I(X;Y) = H(X) + H(Y) - H(X,Y)$$

  and in particular

$$I(X;Y) = I(Y;X)$$

  which also implies

$$I(X;Y) = H(X) - H(X|Y).$$

  - Intuition: X and Y reveal an equal amount of information about each other (or maybe this is not that intuitive!)
  - Proof: We have from the above alternative form that

$$\begin{aligned} I(X;Y) &= \mathbb{E}\left[\log_2 \frac{P_{XY}(X,Y)}{P_X(X)P_Y(Y)} \right] \\ &= \mathbb{E}\left[\log_2 \frac{1}{P_X(X)} + \log_2 \frac{1}{P_Y(Y)} + \log_2 P_{XY}(X,Y) \right] \\ &= H(X) + H(Y) - H(X,Y), \end{aligned}$$

    where we first expanded the logarithm and then applied the definition of (joint) entropy.

- **Non-negativity:** $I(X;Y) \geq 0$ with equality if and only if X and Y are independent.

  - Intuition: One random variable cannot tell us a "negative amount" of information about the other.
  - Proof: Using the above-established identity $I(X;Y) = D(P_{XY} \| P_X \times P_Y)$, this is just a special case of $D(P\|Q) \geq 0$ with equality if and only if $P = Q$.

- **Upper bounds:** We have

$$I(X;Y) \le H(X) \le \log_2 |\mathcal{X}|$$
$$I(X;Y) \le H(Y) \le \log_2 |\mathcal{Y}|.$$

  - Intuition: The information X reveals about Y (mutual information) is at most the prior information in X (entropy).
  - Proof: To show that $I(X;Y) \le H(X)$, combine $I(X;Y) = H(X) - H(X|Y)$ (see above) and $H(X|Y) \ge 0$ (conditional or unconditional entropy is never negative). We already showed $H(X) \le \log_2 |\mathcal{X}|$ earlier, and the remaining claims follow by symmetry, reversing the roles of X and Y.

- **Chain rule:**

$$I(X_1, \ldots, X_n; Y) = \sum_{i=1}^{n} I(X_i; Y | X_1, \ldots, X_{i-1}).$$

  - Intuition: Similar to the chain rule for entropy.
  - Proof: Write $I(X_1, \ldots, X_n; Y) = H(X_1, \ldots, X_n) - H(X_1, \ldots, X_n|Y)$ and apply the chain rule for entropy to both terms.

- **Data processing inequality:** If Z depends on (X, Y) only through Y (often stated via the terminology "$X \to Y \to Z$ forms a Markov chain", and equivalent to the statement "X and Z are conditionally independent given Y"), then

$$I(X;Z) \le I(X;Y).$$

  - Intuition: Processing Y (to produce Z) cannot increase the information available regarding X.
  - Proof: As stated above, the statement "Z depends on (X, Y) only through Y" is equivalent to "Z and X are conditionally independent given Y". This means that the property $P_{Z|XY} = P_{Z|Y}$ (as assumed in the result) is equivalent to $P_{X|YZ} = P_{X|Y}$. To deduce the result, we write

$$\begin{aligned} I(X;Z) &\overset{(a)}{=} H(X) - H(X|Z) && (3)\\ &\overset{(b)}{\le} H(X) - H(X|Y,Z) && (4)\\ &\overset{(c)}{=} H(X) - H(X|Y) && (5)\\ &\overset{(d)}{=} I(X;Y), && (6) \end{aligned}$$

    where (a) and (d) use the definition of mutual information, (b) follows since conditioning reduces entropy, and (c) holds because $H(X|Y,Z) = \mathbb{E}\big[\log \frac{1}{P_{X|YZ}(X|Y,Z)}\big] = \mathbb{E}\big[\log \frac{1}{P_{X|Y}(X|Y)}\big] = H(X|Y)$ by the above-established fact $P_{X|YZ} = P_{X|Y}$.

- Variations: (See the tutorial)
  - If $X \to Y \to Z$ then $I(X;Z) \leq I(Y;Z)$.
  - If $W \to X \to Y \to Z$ then $I(W;Z) \leq I(X;Y)$.

- **Partial sub-additivity:** If $(Y_1, \ldots, Y_n)$ are conditionally independent given $(X_1, \ldots, X_n)$, and in addition Y_i depends on $(X_1, \ldots, X_n)$ only through X_i, then

$$I(X_1, \ldots, X_n; Y_1, \ldots, Y_n) \leq \sum_{i=1}^{n} I(X_i; Y_i).$$

  However, without the conditional independence assumptions, this property may fail to hold. This will be proved in a later tutorial and will be important when we get to the topic of channel coding.

8 (Optional) Entropy of English Text

- Shannon's famous 1948 paper discussed several (intentionally over-simplified) probabilistic models for generating English text; see Figure 1 below.

- Stated differently, #3 generates each letter conditioned on the previous one, #4 conditions on the previous two, #5 lets the "alphabet" $\mathcal{X}$ be the set of all words rather than the set of all characters and generates each word independently, and #6 generates each word conditioned on the previous one.

- **Fundamental question:** How much information (entropy) does each letter of English text tell us?
  - The entropy $H(X)$ of a single character doesn't capture the fact that previous characters help in predicting the next one.
  - As detailed in Chapter 4 of Cover/Thomas, a more meaningful measure in such scenarios is
$$H(X_n | X_1, \ldots, X_{n-1}),$$
    representing the uncertainty of a given character given all of the previous ones.
  - Fitting a model to English text and then calculating the entropy of that model is prone to be inaccurate (too complex to fit a very accurate model!) – is there a simpler approach?

- **Key idea:** The entropy is closely related to *how many guesses are needed (on average) before the correct character is guessed*, by an optimal guessing algorithm.
  - Intuitively, entropy is a measure of uncertainty, and higher uncertainty means more guesses will be needed on average.

1. Zero-order approximation (symbols independent and equiprobable).

   XFOML RXKHRJFFJUJ ZLPWCFWKCYJ FFJEYVKCQSGHYD QPAAMKBZAACIBZL-HJQD.

2. First-order approximation (symbols independent but with frequencies of English text).

   OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL.

3. Second-order approximation (digram structure as in English).

   ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TU-COOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE.

4. Third-order approximation (trigram structure as in English).

   IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONS-TURES OF THE REPTAGIN IS REGOACTIONA OF CRE.

5. First-order word approximation. Rather than continue with tetragram, ... , n-gram structure it is easier and better to jump at this point to word units. Here words are chosen independently but with their appropriate frequencies.

   REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NAT-URAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE.

6. Second-order word approximation. The word transition probabilities are correct but no further struc-ture is included.

   THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHAR-ACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED.

Figure 1: Excerpt from Shannon's paper.

  - Writing an optimal guessing algorithm is hard (though an interesting machine learning problem!), so experiments were done under the assumption that *humans are near-optimal guessers.*
  - Using some theory behind the "optimal guessing" viewpoint, and observing the average number of guesses that several humans required, it was estimated that the entropy of English text is only around **1.34 bits per character**
  - Much less than the highest possible value of $\log_2 27 \approx 4.75$ with 27 characters! (a-z and "space")

- See Chapter 6 of Cover/Thomas for further details.

9 (Optional) Other Properties

More properties of entropy.

- **Functions of random variables:** For a deterministic function f, we have
$$H(f(X)) \leq H(X)$$
  - Intuition: Transforming a random variable doesn't increase its information content.
  - Proof: Since $f(X)$ is deterministic given X, we have $H(f(X)|X) = 0$, and hence
$$\begin{aligned} H(X) &= H(X) + H(f(X)|X) \\ &= H(X, f(X)) \\ &= H(f(X)) + H(X|f(X)) \\ &\geq H(f(X)), \end{aligned}$$
    where the first and third lines use the chain rule, and the last line uses non-negativity.
  - An alternative proof is explored in the tutorial.
- **Information-preserving transform:** If Y depends on X only through $f(X)$, then
$$H(Y|X) = H(Y|f(X)).$$
  - Intuition: By the assumption, $f(X)$ already gives us all that X can tell us about Y.
  - Proof: Let $F = f(X)$. By assumption $P_{Y|X}(y|x) = P_{Y|F}(y|f(x))$ for some $P_{Y|F}$, and hence
$$H(Y|X) = \mathbb{E}\left[\log_2 \frac{1}{P_{Y|X}(Y|X)}\right] = \mathbb{E}\left[\log_2 \frac{1}{P_{Y|F}(Y|f(X))}\right] = H(Y|f(X)).$$
- **Concavity:** The entropy $H(X)$ satisfies a useful property known as concavity (as a function of the distribution P_X).

More properties of mutual information.

- **Functions of random variables.** If Y depends on X only through $f(X)$, then
$$I(X;Y) = I(f(X);Y).$$
  This follows easily from the analogous conditional entropy property above upon applying $I(X;Y) = H(Y) - H(Y|X)$.
- **Convexity properties:** Mutual information $I(X;Y)$ is concave in P_X for fixed $P_{Y|X}$, and is convex in $P_{Y|X}$ for fixed P_X.
  - For an introduction to convexity, see the book "Convex Optimization" by Boyd and Vandenberghe

Lecture 4: Symbol-Wise Source Coding

Useful references:

- Cover/Thomas Chapter 5
- MacKay Chapter 5

10 Symbol-Wise Coding

Setup.

- Consider a discrete random variable X with probability mass function (PMF) P_X.
  - For example, for text (without spaces/punctuation) X might take one of 26 characters $\{a, \ldots, z\}$, with $P_X(e)$ being highest, $P_X(q)$ being low, etc.
  - More generally, the set of all symbols is denoted by $\mathcal{X}$ (called the "alphabet" even when not referring to the English alphabet or even text).
- Symbol-wise source coding maps each $x \in \mathcal{X}$ to some binary sequence $C(x)$. The length of this sequence is denoted by $\ell(x)$.
  - e.g., Map 'x' to 0011010 and 'q' to 0010100 because they are uncommon symbols, map 'e' to 1 because it is a very common symbol.

  Since having a small length is so fundamental, we provide the following formal definition.
- **Definition.** The average length of a code $C(\cdot)$ is given by

$$L(C) = \sum_{x \in \mathcal{X}} P_X(x)\ell(x).$$

- We need to be able to map back from the binary sequences to the original alphabet, so we cannot make every binary sequence short!

- Let's look at the decoding conditions we would like to have.

Decodability conditions.

- To have any hope of mapping the binary sequences back to the original alphabet, we need that $C(x) \neq C(x')$ whenever $x \neq x'$. This condition is so trivial that it doesn't really need a name, but sometimes it's called the *nonsingular* property.
- We are actually interested in coding multiple alphabet symbols in succession, so being nonsingular is not enough. Consider the following code for $\mathcal{X} = \{1, 2, 3, 4\}$:

  a → 0
  b → 1
  c → 00
  d → 11

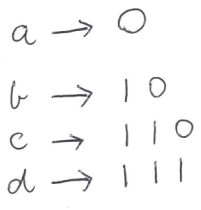
  Clearly, there is no way to distinguish the sequence 'aabb' from 'cd'.
- **Definition.** A code $C(\cdot)$ is said to be *uniquely decodable* if no two sequences (of equal or differing lengths) of symbols in $\mathcal{X}$ are coded to the same concatenated binary sequence. That is, $x_1, \ldots, x_n$ can always uniquely be identified from the string $C(x_1) \ldots C(x_n)$.
- **Example.** The following code is uniquely decodable:

  a → 1
  b → 10
  c → 100
  d → 1000

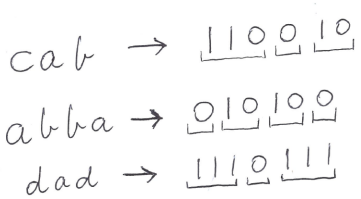
  While unique decodability is easy to see in this example, it can be tricky to verify in larger codes. It is more convenient to work with the following *seemingly* more restrictive condition.
- **Definition.** A code $C(\cdot)$ is said to be *prefix-free* if no codeword is a prefix of any other (i.e., it is not possible to append more bits to some $C(x)$ in order to produce some other $C(x')$).
  - Sometimes the terminology *instantaneous code* is used to mean the same thing.
  - It turns out (we will omit the proof) that restricting to prefix-free codes instead of general uniquely decodable codes does not lose us anything in the average length we can achieve.
  - However, while decoding uniquely decodable codes is challenging in general, decoding prefix-free codes is trivial: As soon as the binary sequence matches some $C(x)$, output x, and then iteratively continue with the rest of the binary sequence.

- We will therefore focus primarily on prefix-free codes.

- **Example.** The following code is prefix-free:



  Here is a simple example of decoding an encoded sequence for this code:



- **Note:** We will focus on binary codes (which are by far the most common), but the vast majority of what we will cover can be easily extended to D-ary codes for $D > 2$ (e.g., for $D = 3$, we have ternary codes with digits $\{0, 1, 2\}$, so we can have mappings like $b \to 21$, $q \to 0121$, $z \to 2222$).

11 Kraft's Inequality and the Entropy Bound

Kraft's Inequality.

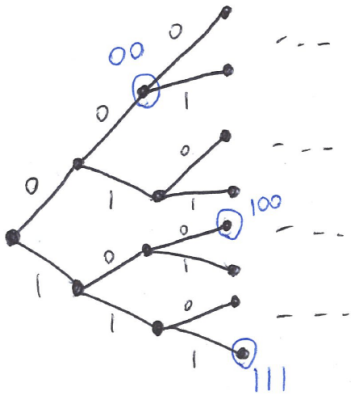
- Clearly not all possible combinations of lengths are possible, e.g., we cannot map every letter $a \ldots z$ to a sequence of length 3 or less – there are not enough such sequences. Even when there are enough sequences, not all allocations of symbols to sequences will be prefix-free (or uniquely decodable). Kraft's inequality gives a useful condition that any prefix-free code must satisfy.

- **Theorem (Kraft's inequality).** Any prefix-free code $C(\cdot)$ that maps each $x \in \mathcal{X}$ to a codeword of length $\ell(x)$ must satisfy

$$\sum_{x \in \mathcal{X}} 2^{-\ell(x)} \leq 1.$$

- **Proof**:

  - Represent the codewords by a binary tree as follows:



- By the prefix-tree assumption, if there is a codeword at some point in the tree, there are no codewords further down the tree.
- Now, consider starting at the root and then repeatedly branching either way with probability $\frac{1}{2}$ each until a codeword is hit.
- Clearly, the probability of a given length-$\ell(x)$ codeword being hit is exactly $2^{-\ell(x)}$.
- But the total probability of hitting codewords cannot exceed one, so summing up all the probabilities gives $\sum_{x\in\mathcal{X}} 2^{-\ell(x)} \leq 1$.

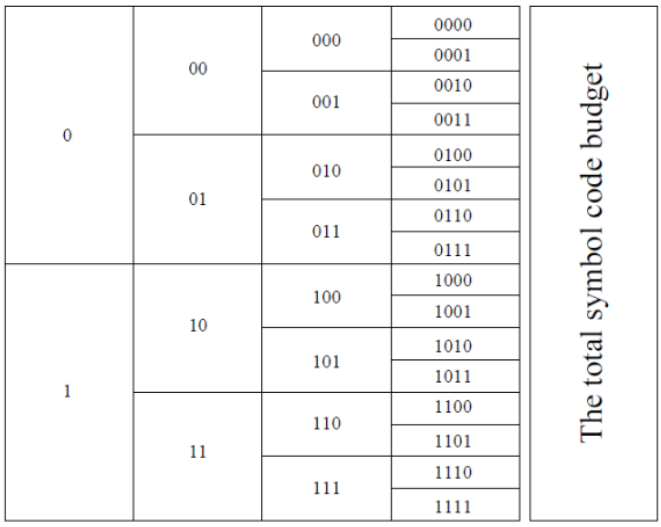
- **Theorem (Existence property).** If a given set of integers $\{\ell(x)\}_{x\in\mathcal{X}}$ satisfies $\sum_{x\in\mathcal{X}} 2^{-\ell(x)} \leq 1$, then it is possible to construct a prefix-free code that maps each $x \in \mathcal{X}$ to a codeword of length $\ell(x)$.
  - Proof outline: Essentially proved by choosing codewords of a suitable length on a tree like the one shown above, starting with those having the smallest length. When $\sum_{x\in\mathcal{X}} 2^{-\ell(x)} \leq 1$, we never "run out of space" on the tree.
  - Hence, the condition in Kraft's inequality is both necessary and sufficient for the existence of a prefix-free code having such lengths.

- **Proof:**

  For notational convenience, let $\mathcal{X} = \{1, \ldots, N\}$, and denote the corresponding lengths by $\{\ell_1, \ldots, \ell_N\}$.

  Think of the codewords illustrated in Figure 1 as being in a 'codeword supermarket', with size indicating cost. We imagine purchasing codewords one at a time, starting from the shortest codewords (i.e., the biggest purchases), using the budget shown at the right of Figure 1. We start at one side of the codeword supermarket, say the top, and purchase the first codeword of the required length ℓ. We advance down the supermarket a distance $2^{-\ell}$, and purchase the next codeword of the next required length, and so forth.

  Because the codeword lengths are getting longer, and the corresponding intervals are getting shorter, we can always buy an adjacent codeword to the latest purchase, so



The codeword supermarket and the symbol coding budget. The 'cost' 2^{-l} of each codeword (with length l) is indicated by the size of the box it is written in. The total budget available when making a uniquely decodeable code is 1.

Figure 2: Illustration of the "supermarket budget".

there is no wasting of the budget. Thus at the I'th codeword we have advanced a distance $\sum_{i=1}^{I} 2^{-\ell_i}$ down the supermarket; if $\sum_{i=1}^{I} 2^{-\ell_i} \leq 1$, we will have purchased all the codewords without running out of budget.

Entropy bound.

- **Theorem.** For $X \sim P_X$ and any prefix-free code $C(\cdot)$, the expected length satisfies

$$L(C) \geq H(X),$$

with equality if and only if $P_X(x) = 2^{-\ell(x)}$ for all $x \in \mathcal{X}$.

  - Hence, entropy provides a fundamental limit – we can never get an average length smaller than the entropy using a prefix-free code.
  - Even though we are only stating/proving it for the prefix-free case, it can be shown that the same holds for any uniquely decodable code.

- **Proof.**

  - Recall the definition of KL divergence, $D(P\|Q) = \sum_x P(x) \log_2 \frac{P(x)}{Q_X(x)}$.
  - Observe that

$$\begin{aligned} L(C) - H(X) &\overset{(a)}{=} \sum_x P_X(x)\ell(x) - \sum_x P_X(x) \log_2 \frac{1}{P_X(x)} \\ &\overset{(b)}{=} \sum_x P_X(x) \log_2 2^{\ell(x)} - \sum_x P_X(x) \log_2 \frac{1}{P_X(x)}, \end{aligned}$$

  where (a) is by definition, and (b) simply uses $c = \log_2(2^c)$.

- To simplify notation, let $Z = \sum_{x \in \mathcal{X}} 2^{-\ell(x)}$, and define $Q_X(x) = \frac{2^{-\ell(x)}}{Z}$ which is a valid PMF (i.e., has non-negative values summing to one).
- Re-arranging terms in the definition of $Q_X(x)$ gives $2^{\ell(x)} = \frac{1}{Z \cdot Q_X(x)}$, and substitution into the above equation gives

$$\begin{aligned} L(C) - H(X) &= \sum_x P_X(x) \log_2 \frac{1}{Z Q_X(x)} - \sum_x P(x) \log_2 \frac{1}{P_X(x)} \\ &\overset{(a)}{=} \log_2 \frac{1}{Z} + \sum_x P_X(x) \log_2 \frac{P_X(x)}{Q_X(x)} \\ &\overset{(b)}{=} \log_2 \frac{1}{Z} + D(P_X \| Q) \\ &\overset{(c)}{\geq} 0, \end{aligned}$$

  where (a) uses simple re-arranging (and $\log_2 \frac{1}{\alpha} = -\log_2 \alpha$), (b) uses the definition of KL divergence, and (c) uses $Z \leq 1$ (by Kraft's inequality) and $D(P_X \| Q_X) \geq 0$ (KL divergence between two PMFs is always non-negative).
- To get $L(C) = H(X)$, we need both $Z \leq 1$ and $D(P_X \| Q_X) \geq 0$ to hold with equality. The former condition gives $Q_X(x) = 2^{-\ell(x)}$ (see the definition of Q_X), and the latter gives $P_X = Q_X$ (as established in the previous lecture), so overall we require $P_X(x) = 2^{-\ell(x)}$ for all x.

- **Implication.** If our probabilities all contain powers of two ($\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$, etc.), we can bring the average code length all the way down to the entropy.
  - A simple example:

SYMBOL	PROBABILITY	CODEWORD
a	$\frac{1}{2}$	0
b	$\frac{1}{4}$	10
c	$\frac{1}{8}$	110
d	$\frac{1}{16}$	1110
e	$\frac{1}{16}$	1111

  - Notice that the lengths satisfy $\ell(x) = \log_2 \frac{1}{P_X(x)}$ for all $x \in \{a, b, c, d, e\}$

12 Shannon-Fano Code

- Based on the "...equality if and only if..." statement in the entropy bound theorem, we can think of $\ell^*(x) = \log_2 \frac{1}{P_X(x)}$ as being the "ideal" code length. However, it can only be attained when all values of $\frac{1}{P_X(x)}$ are powers of two.

- The **Shannon-Fano code** simply rounds the ideal lengths up to the nearest integer:

$$\ell(x) = \left\lceil \log_2 \frac{1}{P_X(x)} \right\rceil,$$

where $\lceil \cdot \rceil$ is the ceiling operation (i.e., rounding up).

- These lengths satisfy the conditions of the "Existence property" theorem above, since

$$\sum_{x\in\mathcal{X}} 2^{-\ell(x)} = \sum_{x\in\mathcal{X}} 2^{-\lceil \log_2 \frac{1}{P_X(x)} \rceil} \leq \sum_{x\in\mathcal{X}} 2^{-\log_2 \frac{1}{P_X(x)}} = \sum_{x\in\mathcal{X}} P_X(x) = 1,$$

where we used $\lceil \alpha \rceil \geq \alpha$ and $-\log_2 \frac{1}{\alpha} = \log_2 \alpha$. Hence, that theorem implies that we can indeed construct a prefix-free code with the above lengths.

- **Theorem.** The average length $L(C)$ of the Shannon-Fano code satisfies

$$H(X) \leq L(C) < H(X) + 1,$$

so is within one bit of the best average length possible.

  - Proof: The lower bound is just a repetition of the entropy bound. To prove the upper bound, we use the fact that $\lceil \alpha \rceil < \alpha + 1$ to deduce the following:

$$\begin{aligned} L(C) &= \sum_{x\in\mathcal{X}} P_X(x)\ell(x) \\ &= \sum_{x\in\mathcal{X}} P_X(x) \left\lceil \log_2 \frac{1}{P_X(x)} \right\rceil \\ &< \sum_{x\in\mathcal{X}} P_X(x) \left(\log_2 \frac{1}{P_X(x)} + 1 \right) \\ &= H(X) + 1, \end{aligned}$$

    where the last step uses the definition of entropy and $\sum_{x\in\mathcal{X}} P_X(x) = 1$. (Note: By following similar steps but instead applying $\lceil \alpha \rceil \geq \alpha$, we get an alternative proof of the lower bound.)

- While the addition of at most 1 bit may seem innocuous, it can be very significant (e.g., for a "low-information" source, maybe $H(X)$ itself is only 0.5 bits!)

- **Theorem (Mismatched case).** If the true distribution is P_X but the lengths are chosen according to Q_X (i.e., $\ell(x) = \lceil \log_2 \frac{1}{Q_X(x)} \rceil$), then the Shannon-Fano code satisfies

$$H(X) + D(P_X \| Q_X) \leq L(C) < H(X) + D(P_X \| Q_X) + 1.$$

  - Proof: Similar to above, also using $\mathbb{E}_P\big[\log_2 \frac{1}{Q_X(X)}\big] = \mathbb{E}_P\big[\log_2 \frac{P_X(X)}{Q_X(X)P_X(X)}\big] = \mathbb{E}_P\big[\log_2 \frac{1}{P_X(X)} + \log_2 \frac{P_X(X)}{Q_X(X)}\big] = H(X) + D(P_X\|Q_X)$.
  - Hence, if an inaccurate distribution is used (not-so-small $D(P_X\|Q_X)$) then the penalty due to mismatch may also be significant.

13 Huffman Code

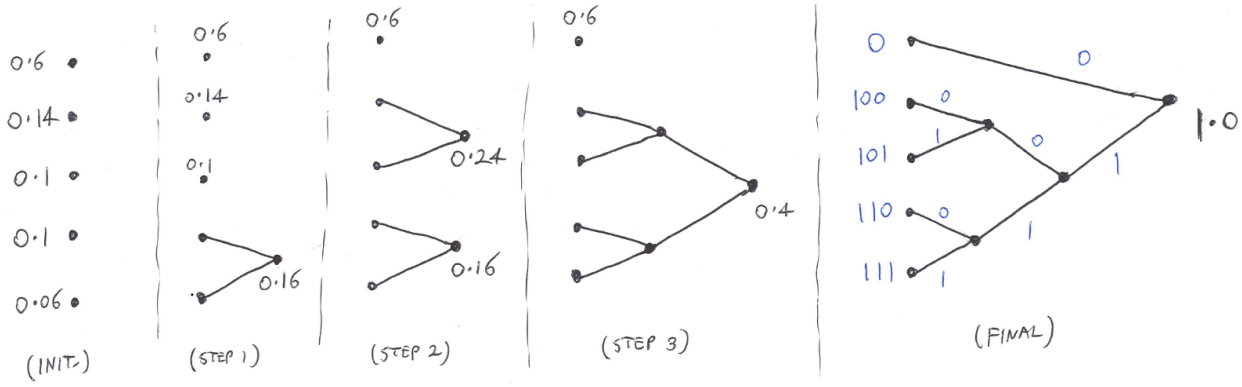
- At this stage it is natural to ask whether it is possible to find the *optimal* symbol code, in the sense of minimizing $L(C)$ while being uniquely decodable. This remained a seemingly challenging open problem until an *extremely simple* solution was given by Huffman (as part of a homework question!).

- **Huffman code.** Construct a tree as follows:

  - List the symbols of $\mathcal{X}$ from highest probability to lowest.
  - Draw a branch connecting the two symbols with the lowest probability and label the merged point with the sum of the two associated probabilities.
  - Repeat the previous step (with the two original probabilities replaced by the merged probability) until everything has merged to a single point with a total probability of 1.

  Once this tree is constructed, we label the two edges in each branch as 0 and 1, and then let the codewords be the labels encountered when traversing from the end back to the start.

- An illustration:



- **Theorem.** No uniquely decodable symbol code can achieve a smaller average length $L(C)$ than the Huffman code.
  - Since Huffman coding is at least as good as Shannon-Fano coding, it also satisfies the average length bound $H(X) \leq L(C) < H(X) + 1$.

- **Proof:**

  Let the probability distribution of random variable X over an alphabet $\mathcal{X}$ be $P_X = \{p_1, p_2, ..., p_N\}$, where $N = |\mathcal{X}|$ and $p_1 \leq p_2 \leq \ldots \leq p_N$. There exists an optimal prefix-free code $C(X)$ with codeword lengths $\{\ell_1, \ell_2, ..., \ell_N\}$ such that:

  1. $\ell_1 \geq \ell_2 \geq .... \geq \ell_N$.
     Suppose there exists an optimal prefix-free code $C'(X)$ with codeword lengths $\{\ell_1, \ell_2, ..., \ell_N\}$ that do not satisfy $\ell_1 \geq \ell_2 \geq .... \geq \ell_N$. Then there exists $\ell_i < \ell_j$ for some $i < j$ and $i, j \in \{1, 2, 3, .., N\}$. Let x_i and x_j be the corresponding symbols. Consider exchanging the codewords of x_i and x_j in $C'(X)$ and call the new prefix-free code $C''(X)$. Since $i < j$ which implies $p_i \leq p_j$, and therefore $p_i(\ell_j - \ell_i) \leq p_j(\ell_j - \ell_i)$. Hence, $p_i\ell_j + p_j\ell_i \leq p_i\ell_i + p_j\ell_j$ which implies that $L(C'', X) \leq L(C', X)$. Hence $C''(X)$ is also an optimal prefix-free code. Continue similarly till the condition is satisfied.
  2. $\ell_1 = \ell_2$.
     Suppose there exists an optimal prefix-free code $C(X)$ satisfying $\ell_1 > \ell_2 \geq .... \geq \ell_N$.. Since C' is prefix-free, if we remove the final $\ell_1 - \ell_2$ bits from the longest codeword, we must still have a prefix-free code (since those removed bits have no impact on the answer to the question "is [some other codeword] a prefix of [this codeword]?"). Let $C'(X)$ be the resulting modified code. It is easy to see that $L(C', X) < L(C, X)$ contradicting the optimality of the prefix-free code $C(X)$. Thus $\ell_1 = \ell_2 \geq .... \geq \ell_N$.

Let the Huffman code for P_X have lengths $\{\ell'_1, \ell'_2, ..., \ell'_N\}$. Note that by construction $\ell'_1 = \ell'_2 \geq \ell'_3 .... \geq \ell'_N$. Also $\{\ell'_1 - 1, \ell'_3, \ell'_4, \ldots, \ell'_N\}$ are the lengths of the Huffman code for $P_{\hat{X}} = \{p_1 + p_2, p_3, \ldots, p_N\}$.

We show $\sum_{i=1}^{N} p_i \ell_i \geq \sum_{i=1}^{N} p_i \ell'_i$ by induction on N.

Base-case: $N = 2$ is clear.

For general N, consider,

$$\begin{aligned}
\sum_{i=1}^{N} p_i \ell_i &= (p_1 + p_2)\ell_1 + \sum_{i=3}^{N} p_i \ell_i \\
&= (p_1 + p_2) + (p_1 + p_2)(\ell_1 - 1) + \sum_{i=3}^{N} p_i \ell_i \\
&\geq (p_1 + p_2) + (p_1 + p_2)(\ell'_1 - 1) + \sum_{i=3}^{N} p_i \ell'_i \qquad \text{(from induction hypothesis)} \\
&= (p_1 + p_2)\ell'_1 + \sum_{i=3}^{N} p_i \ell'_i. \\
&= \sum_{i=1}^{N} p_i \ell'_i.
\end{aligned}$$

14 Shared-randomness assisted code

Let X be a random variable with probability distribution $P_X = \{p_1, p_2, p_3, ..., p_N\}$ over alphabet $A_X = \{x_1, x_2, x_3, ..., x_N\}$. Alice gets a symbol $X = x_k$ (with probability p_k) and she wants to communicate it to Bob. Let Alice and Bob share randomness as shown in Figure 3, that is infinitely many X^i each independent and identically distributed as X.

Consider the following protocol:

1. Alice sends to Bob the smallest index j such that $X^j = x_k$ (using a prefix free code to encode j using $\lceil \log j + 2 \log \log j + 1 \rceil$ bits).
2. Bob outputs X^j.

Let J_k be the random variable representing the smallest index j communicated by Alice when $X = x_k$. We have $\Pr[J_k = r] = p_k(1 - p_k)^{r-1}$. Hence,

$$\lambda_k := \mathbb{E}[J_k] = \sum_{r=1}^{\infty} \Pr[J_k = r] \cdot r = \sum_{r=1}^{\infty} r p_k (1 - p_k)^{r-1}.$$

We have,

$$\lambda_k (1 - p_k) = \sum_{r=1}^{\infty} r p_k (1 - p_k)^{r} = \sum_{r=1}^{\infty} (r - 1) p_k (1 - p_k)^{r-1}.$$

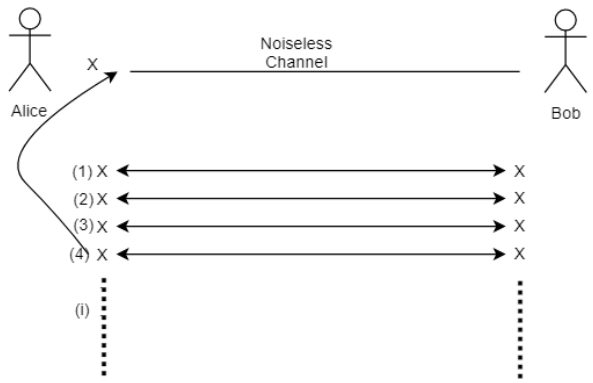


Figure 3: Communication using shared randomness

By subtracting the above two equations, we get

$$\lambda_k p_k \;=\; \sum_{r=1}^{\infty} p_k(1-p_k)^{r-1} \;=\; \frac{p_k}{1-(1-p_k)} \;=\; \frac{p_k}{p_k} \;=\; 1.$$

Hence expected communication from Alice to Bob (in bits) is:

$$\begin{aligned}
&\sum_k p_k \sum_{j=1}^{\infty} \Pr[J_k = j]\left(\lceil \log j + 2\log\log j + 1\rceil\right) \\
&\leq 2 + \sum_k p_k \sum_{j=1}^{\infty} \Pr[J_k = j](\log j + 2\log\log j) \\
&\leq 2 + \sum_k p_k \left(\log\left(\sum_{j=1}^{\infty} \Pr[J_k = j] j\right) + 2\log\log\left(\sum_{j=1}^{\infty} \Pr[J_k = j] j\right)\right) \quad \text{(concavity of log and loglog)} \\
&= 2 + \sum_k p_k \left(\log\frac{1}{p_k} + 2\log\log\frac{1}{p_k}\right) \\
&\leq 2 + H(X) + 2\log\sum_k p_k \log\frac{1}{p_k} \quad \text{(concavity of log)} \\
&= H(X) + 2\log H(X) + 2.
\end{aligned}$$

15 Discussion

- Perhaps the most significant limitation of symbol codes is that they do not exploit *memory* (i.e., dependence between subsequent symbols). For instance, given a 'q' in English the next character is very likely to be 'u', or to go even further, we can be very confident about the next character in the sequence "Fill in the blan_".
- However, we can extend symbol codes to exploit such dependencies as follows:
  - Instead of coding one character at a time, group them into chunks of a given length (say, 5 symbols) and do Huffman coding with $\mathcal{X}^5$ in place of $\mathcal{X}$, and $P_{X_1X_2X_3X_4X_5}$ in place of P_X.
  - Advantage 1: We can exploit the fact that, for example, "apple" is a much more probable sequence than "ealpp" despite containing the same characters.
  - Advantage 2: Even with independent symbols (e.g., $P_{X_1X_2X_3X_4X_5} = \prod_{i=1}^{5} P_X(x_i)$) we can improve the "one bit extra" guarantee. More generally letting $N = 5$ denote the length we are coding over, the standard one-bit guarantee gives the following under independent symbols:

    $$NH(X) \leq L(C) \leq NH(X) + 1,$$

    since $H(X_1, \ldots, X_N) = \sum_{i=1}^{N} H(X_i)$ for independent symbols. But by dividing by N, this gives

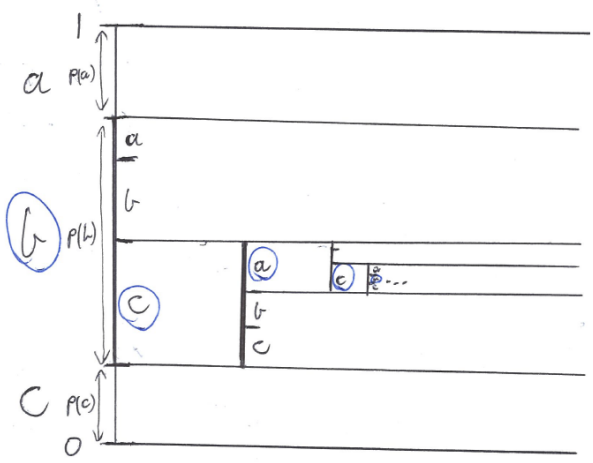
    $$H(X) \leq \text{Average length per symbol} \leq H(X) + \frac{1}{N},$$

    so that the loss of 1 bit in the upper bound is improved to $\frac{1}{N}$.
  - Disadvantage 1: Determining the distribution $P_{X_1,\ldots,X_N}$ accurately is very difficult.
  - Disadvantage 2: Even if the joint distribution is known, sorting $|\mathcal{X}|^N$ probabilities in the Huffman coding algorithm becomes computationally challenging even for moderate values of N. The complexity increases exponentially with N.

16 (Optional) Beyond Symbol-Wise Codes

Arithmetic codes

- Arithmetic codes are a very elegant technique for sequentially coding sources with memory when the conditional distribution $P_{X_i|X_1,\ldots,X_{i-1}}$ is known. The idea is illustrated in the following:



- For concreteness, suppose the alphabet is 'a', 'b', 'c'.
- Start with an interval ranging from 0 to 1.
- Split the interval into three regions of width $P_{X_1}(a)$, $P_{X_1}(b)$, and $P_{X_1}(c)$.
- After observing $X_1 = b$, move into the region of width $P_{X_1}(b)$.
- Split the width-$P_{X_1}(b)$ region into three regions proportional to $P_{X_2|X_1}(a)$, $P_{X_2|X_1}(b)$, and $P_{X_2|X_1}(c)$.
- After observing $X_2 = c$, move into the corresponding sub-region.
- Continue recursively until the entire input sequence has been read.

- At the end of this process, we are left with a very small sub-interval $\mathcal{I}$ of $[0, 1]$. How do we map this to a binary sequence?

- **Key idea.**
  - Every point in the interval $[0, 1]$ corresponds to an infinite binary sequence (e.g., $\frac{1}{3}$ maps to $0.010101\ldots$, $\frac{1}{2}$ maps to $0.10000\ldots$, etc.).
  - A finite-length binary sequence (e.g., 0.010101) then corresponds to an *interval* (e.g. starting from the number represented by 0.010101 followed by infinitely many zeros, ending at the number 0.010101 followed by infinitely many ones).
  - Output a finite-length binary sequence that is just long enough for its corresponding interval to be a sub-interval of $\mathcal{I}$.

- The notion of entropy as a fundamental compression limit can be extended to broad types of sources with memory (see Cover/Thomas Chapter 4), and it can be shown that arithmetic coding can encode a sequence $(x_1, \ldots, x_n)$ down to at most

$$\ell(x_1, \ldots, x_n) \leq \log_2 \frac{1}{P_{X_1,\ldots,X_n}(x_1, \ldots, x_n)} + 2$$

bits, which is within 2 bits of the "ideal length". In particular, we get

$$\text{Average Total Length} \leq H(X_1, \ldots, X_n) + 2,$$

and for a memoryless source, the number of bits per symbol is at most $H(X) + \frac{2}{n}$.

- See Cover/Thomas Section 13.3 or MacKay Section 6.2 for further details.

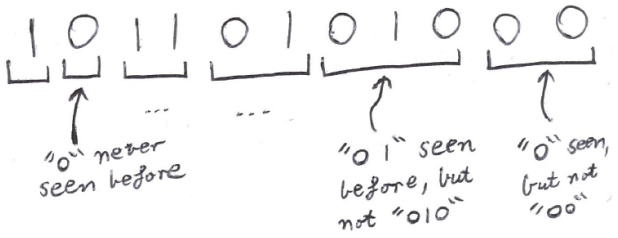
Lempel-Ziv code

- A disadvantage of arithmetic codes is the need to know $P_{X_1,\ldots,X_n}$. A class of codes known as *Lempel-Ziv* (LZ) codes are *universal*, in that they do not use any knowledge of the source distribution.

- For memoryless sources, and also broad classes of sources with memory, LZ codes are efficient enough to code almost down to the entropy (albeit with the "second-order" term being $O(\log n)$, which is a fair bit higher than the 2 bits attained by arithmetic coding).

- The encoding can roughly be described as follows:

  - Step 1: Parse the string into substrings that haven't been observed earlier:



  - Step 2: Encode each parsed sub-string into a sequence of bits of the form (pointer, new bit), where "pointer" identifies the index of the sub-string that matches the current one with the final bit removed, and "new bit" describes that final bit.
  - Encoding: Store the sequence of pointers (represented in binary) and new bits.

- See Cover/Thomas Section 13.4 or MacKay Section 6.4 for further details.

Lecture 5: Block Source Coding

Useful references:

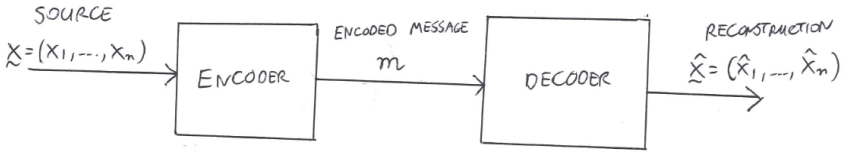
- Cover/Thomas Chapter 3
- MacKay Chapter 4
- Shannon's 1948 paper "A Mathematical Introduction to Communication"

17 Setup

Introduction.

- In the previous lecture, we mapped individual source symbols $x \in \mathcal{X}$ to *variable-length* binary sequences one at a time (symbol coding), and briefly discussed mapping multiple at a time (block coding).
- In this lecture, we consider the following distinct setting:
  - We do not work symbol-by-symbol, but instead apply some encoding function to a *length-n block* $X_1, \ldots, X_n$.
  - The output of the encoder is not a variable-length sequence, but instead an integer $m \in \{1, \ldots, M\}$ for some M. For instance, we might store M on a computer as a *fixed-length* binary sequence of length $\log_2 M$.

  Because each input sequence of length n is mapped to a binary output sequence with length $\log_2 M$, this is sometimes called *fixed-to-fixed* length source coding.
- An illustration:



- At the end of the last lecture, we briefly mentioned variable-length block coding methods, which are in fact more pertinent to practical compression methods. The fixed-length setting, on the other hand, provides a better warm-up for the next lecture's topic of channel coding.

- *Key difference*: Consider the case that the X_i are binary (for simplicity of discussion). If $\log_2 M < n$ (i.e., $M < 2^n$), we clearly can't assign every sequence $(X_1, \ldots, X_n)$ a unique value of m. This means that *some* of the sequences must be decoded incorrectly ("errors").

    - In contrast, in the variable-length setting, we never have an error; some output sequences just come out longer than others.

Formal problem statement.

- The *source* is a sequence $(X_1, \ldots, X_n)$. We focus on *discrete memoryless sources*:

    - Discrete: The alphabet $\mathcal{X}$ is finite;
    - Memoryless: $P_{\mathbf{X}}(\mathbf{x}) = \prod_{i=1}^{n} P_X(x_i)$, i.e., the source symbols are i.i.d. on some distribution P_X. (This is a restrictive assumption, but still an interesting problem to study)

- An *encoder* receives as input a sequence of source symbols $\mathbf{X} = (X_1, \ldots, X_n) \in \mathcal{X}^n$, and maps it to an encoded message $m = f(\mathbf{X})$ in $\{1, \ldots, M\}$.

- A *decoder* receives the encoded message m and maps it to an estimate $\hat{\mathbf{X}} = g(m)$ (in $\mathcal{X}^n$) of the source.

- An error is said to have occurred if $\hat{\mathbf{X}} \neq \mathbf{X}$, and the *error probability* is given by

$$P_{\mathrm{e}} = \mathbb{P}[\hat{\mathbf{X}} \neq \mathbf{X}].$$

- The *rate* is defined to be

$$R = \frac{1}{n} \log_2 M,$$

  and represents the number of bits per source symbol used to represent the encoded value m. The lower the rate, the more we have compressed the source sequence.

A fundamental trade-off.

- Clearly we would like P_{e} to be small.

- We also want M (or equivalently, R) to be small, so that we require less bits to store m.

- The length n also plays a fundamental role and is referred to as the *block length*.

- Key question: What is the fundamental trade-off between error probability P_{e}, rate R, and block length n? In particular, how low can the rate be while keeping the error probability small?

- **Fixed-Length Source Coding Theorem.** For any discrete memoryless source with per-symbol distribution P_X, we have the following:

  - (Achievability) If $R > H(X)$, then for any $\delta > 0$, there exists a (sufficiently large) block length n and a source code (i.e., encoder and decoder) of rate R such that $P_{\mathrm{e}} \leq \delta$;
  - (Converse) If $R < H(X)$, then there exists $\delta > 0$ such every source code of rate R has $P_{\mathrm{e}} > \delta$, regardless of the block length (i.e., P_{e} cannot be arbitrarily small).

  The proofs are respectively given in the next two sections.

18 Typical Sequences and the Asymptotic Equipartition Property

Definition.

- Recall that $\mathbf{X} = (X_1, \ldots, X_n)$ is an i.i.d. sequence with each symbol distributed according to P_X. In the following, let $P_{\mathbf{X}}(\mathbf{x}) = \prod_{i=1}^{n} P_X(x_i)$ be the PMF of $\mathbf{X}$.

- The *typical set* is defined as

$$\mathcal{T}_n(\epsilon) = \left\{ \mathbf{x} \in \mathcal{X}^n \,:\, 2^{-n(H(X)+\epsilon)} \leq P_{\mathbf{X}}(\mathbf{x}) \leq 2^{-n(H(X)-\epsilon)} \right\},$$

  where $\epsilon > 0$ is a fixed (small) constant.

- As we will see shortly, it is called the typical set because for $\mathbf{X} \sim P_{\mathbf{X}}$ the probability that $\mathbf{X} \in \mathcal{T}_n(\epsilon)$ is very close to one.

- After analyzing its properties, we will give some intuition as to how one might have come up with this definition "from scratch".

Properties.

- For any fixed $\epsilon > 0$, four key properties of the typical set are as follows (proofs below):

  1. (Equivalent definition) We have $\mathbf{x} \in \mathcal{T}_n(\epsilon)$ if and only if

$$H(X) - \epsilon \leq \frac{1}{n} \sum_{i=1}^{n} \log_2 \frac{1}{P_X(x_i)} \leq H(X) + \epsilon$$

     where x_i is the i-th entry of $\mathbf{x}$.

2. (High probability) $\mathbb{P}[\mathbf{X} \in \mathcal{T}_n(\epsilon)] \to 1$ as $n \to \infty$.
3. (Cardinality upper bound) $|\mathcal{T}_n(\epsilon)| \leq 2^{n(H(X)+\epsilon)}$.
4. (Cardinality lower bound) $|\mathcal{T}_n(\epsilon)| \geq (1 - o(1))2^{n(H(X)-\epsilon)}$, where $o(1)$ represents a term that vanishes as $n \to \infty$.

- Interpretation: With high probability (second property), a randomly drawn i.i.d. sequence $\mathbf{X}$ will be one of roughly $2^{nH(X)}$ sequences (third and fourth properties), each of which has probability roughly $2^{-nH(X)}$ (definition of typical set).
    - We call this the *Asymptotic Equipartition Property*, because it states that asymptotically (as $n \to \infty$) the distribution is roughly uniform over $\mathcal{T}_n(\epsilon)$.
- Proofs:
    1. Apply $\frac{1}{n}\log_2(\cdot)$ to the left, middle, and right of the condition $2^{-n(H(X)+\epsilon)} \leq P_{\mathbf{X}}(\mathbf{x}) \leq 2^{-n(H(X)-\epsilon)}$ defining $\mathcal{T}_n(\epsilon)$. The left and right clearly become $H(X) - \epsilon$ and $H(X) + \epsilon$ (since $\log_2(2^\alpha) = \alpha$), and the middle becomes $\frac{1}{n}\log_2 P_{\mathbf{X}}(\mathbf{x}) = \frac{1}{n}\log_2 \prod_{i=1}^n P_X(x_i) = \frac{1}{n}\sum_{i=1}^n \log_2 \frac{1}{P_X(x_i)}$.
    2. Since $\mathbf{X}$ is an i.i.d. sequence, $\frac{1}{n}\sum_{i=1}^n \log_2 \frac{1}{P_X(X_i)}$ is an i.i.d. sum of random variables. The mean of each such random variable is $\mathbb{E}\big[\log_2 \frac{1}{P_X(X)}\big] = H(X)$. Therefore, due to property 1, we see that property 2 simply follows from the law of large numbers.[6]
    3. By the definition of the typical set, if $\mathbf{x} \in \mathcal{T}_n(\epsilon)$ then $P_{\mathbf{X}}(\mathbf{x}) \geq 2^{-n(H(X)+\epsilon)}$. Since any probability is at most one, we have
$$\begin{aligned} 1 &\geq \mathbb{P}[\mathbf{X} \in \mathcal{T}_n(\epsilon)] \\ &= \sum_{\mathbf{x} \in \mathcal{T}_n(\epsilon)} P_{\mathbf{X}}(\mathbf{x}) \\ &\geq \sum_{\mathbf{x} \in \mathcal{T}_n(\epsilon)} 2^{-n(H(X)+\epsilon)} \\ &= |\mathcal{T}_n(\epsilon)| \cdot 2^{-n(H(X)+\epsilon)}. \end{aligned}$$
       Re-arranging gives the third property.
    4. By the definition of the typical set, if $\mathbf{x} \in \mathcal{T}_n(\epsilon)$ then $P_{\mathbf{X}}(\mathbf{x}) \leq 2^{-n(H(X)-\epsilon)}$. Writing

[6]The *law of large numbers* states that the average of n i.i.d. random variables is arbitrarily close to its mean with probability approaching one. See the prerequisite material document for a more formal statement.

property 2 as $\mathbb{P}[\mathbf{X} \in \mathcal{T}_n(\epsilon)] = 1 - o(1)$, we obtain

$$\begin{aligned} 1 - o(1) &= \mathbb{P}[\mathbf{X} \in \mathcal{T}_n(\epsilon)] \\ &= \sum_{\mathbf{x} \in \mathcal{T}_n(\epsilon)} P_{\mathbf{X}}(\mathbf{x}) \\ &\leq \sum_{\mathbf{x} \in \mathcal{T}_n(\epsilon)} 2^{-n(H(X)-\epsilon)} \\ &= |\mathcal{T}_n(\epsilon)| \cdot 2^{-n(H(X)-\epsilon)}. \end{aligned}$$

Re-arranging gives the fourth property.

Implication.

- The above suggests a very simple source coding scheme:
  - Map each typical sequence to a unique integer in $\{1, \ldots, M-1\}$;
  - Map each non-typical sequence to a "dummy value" M.

  The decoder lets $\hat{\mathbf{X}}$ be arbitrary for $m = M$, whereas for $m < M$ it simply outputs the corresponding typical sequence.
- Clearly this scheme is possible if $M = |\mathcal{T}_n(\epsilon)| + 1$, and yields error probability $P_{\mathrm{e}} \leq \mathbb{P}[\mathbf{X} \notin \mathcal{T}_X]$, which is arbitrarily small for sufficiently large n by property 2 above.
- Substituting property 3 above gives $M = 2^{n(H(X)+\epsilon)} + 1$. Since ϵ may be arbitrarily small and the rate is $\frac{1}{n} \log_2 M$, we deduce that *we can get arbitrarily small error probability with a rate arbitrarily close to* $H(X)$.
  - Note that this only holds as $n \to \infty$. The closer we take the rate to $H(X)$ (smaller ϵ) and the smaller we take the error probability, the higher we might need to take the block length n.

A possible thought process behind deriving $\mathcal{T}_n(\epsilon)$.

- Since we only have a finite number of messages $\{1, \ldots, M\}$ to work with, it makes sense to assign them only to the most probable sequences, i.e., those such that

  $$P_{\mathbf{X}}(\mathbf{x}) \geq \gamma$$

  for some $\gamma > 0$. How high can we make γ while still ensuring the set has a high probability?
- After staring at this for a while, one becomes tempted to take the log (to simplify the product in $P_{\mathbf{X}}(\mathbf{x}) = \prod_{i=1}^{n} P_X(x_i)$) to get the equivalent condition

  $$\sum_{i=1}^{n} \log_2 P_X(x_i) \geq \log_2 \gamma.$$

- Recognizing $\sum_{i=1}^{n} \log_2 P_X(X_i)$ as a sum of independent random variables with mean $H(X)$, one realizes that $\log_2 \gamma$ should be chosen as roughly $-nH(X)$ by the law of large numbers. With some re-arranging, the original condition $P_{\mathbf{X}}(\mathbf{x}) \geq \gamma$ reduces to $P_{\mathbf{X}}(\mathbf{x}) \gtrsim 2^{-nH(X)}$.

- Since the law of large numbers works for deviations on both sides of the mean, one then realizes that things also work out if we use the two-sided version $\mathcal{T}_n(\epsilon)$.

(Optional) Alternative "one-sided typicality" proof.

- It is, in fact, not hard to see that we could get to the same "R arbitrarily close to $H(X)$" result using a one-sided typicality notion like that in the above thought process. Specifically, consider the definition

$$\mathcal{T}'_n(\epsilon) = \left\{ \mathbf{x} \in \mathcal{X}^n \,:\, P_{\mathbf{X}}(\mathbf{x}) \geq 2^{-n(H(X)+\epsilon)} \right\}.$$

  We still have $\mathbf{X} \in \mathcal{T}'_n(\epsilon)$ with probability approaching one, and the same upper bound on the total number of sequences satisfying it (by the same proofs as above).

- This variation arguably makes *more sense*, as it encodes all sequences whose value of $P_{\mathbf{X}}(\mathbf{x})$ is sufficiently high – it seems strange to ignore those that are the most probable!

- Nevertheless, two-sided typicality is more common in information theory proofs, and in certain other settings, it is actually useful for mathematical analysis.

19 Fano's Inequality and a Converse Bound

Motivation.

- The idea behind proving the converse part of the source coding theorem is to consider the mutual information $I(\mathbf{X}; \hat{\mathbf{X}})$ as follows.

- Remember that mutual information is how much one random variable reveals about another. If our estimate $\hat{\mathbf{X}}$ is accurate, then the amount of information that it reveals about $\mathbf{X}$ should be roughly equal to $H(\mathbf{X}) = nH(X)$, the prior uncertainty in $\mathbf{X}$. Since $I(\mathbf{X}; \hat{\mathbf{X}}) = H(\mathbf{X}) - H(\mathbf{X}|\hat{\mathbf{X}})$, this is equivalent to saying that we should have $H(\mathbf{X}|\hat{\mathbf{X}}) \approx 0$.

- However, we also have $I(\mathbf{X}; \hat{\mathbf{X}}) \leq H(\hat{\mathbf{X}}) \leq nR$, since there are only 2^{nR} possible $\hat{\mathbf{X}}$ sequences (and the uniform distribution maximizes entropy and gives entropy equaling the log of the number of values).

- Putting these together, we get that having an accurate estimate requires $R > H(X)$.

- Before making this argument rigorous, we need to introduce a tool for formalizing the fact that accurate estimation implies $H(\mathbf{X}|\hat{\mathbf{X}}) \approx 0$.

Fano's inequality.

- In the following, X denotes a generic random variable (or vector), and $\hat{X}$ can be thought of as any estimate of X. At this stage, these do not need to be thought of as necessarily directly related to the definitions in the previous sections.

- Fano's inequality relates two fundamental quantities:

  - The conditional entropy $H(X|\hat{X})$;
  - The "error probability" $P_{\mathrm{e}} = \mathbb{P}[\hat{X} \neq X]$.

  Intuitively, if $H(X|\hat{X})$ is "large", then $\hat{X}$ does not reveal much information about X, so P_{e} must not be too small either (if it were very small, then knowing $\hat{X}$ would tell us a lot about X!).

  Similarly, if P_{e} is small then $H(X|\hat{X})$ should be small too. As an extreme example, if $P_{\mathrm{e}} = 0$ then $\hat{X} = X$ and therefore $H(X|\hat{X}) = 0$.

- **Claim (Fano's Inequality).** For any discrete random variables X and $\hat{X}$ on a common finite alphabet $\mathcal{X}$, we have

$$H(X|\hat{X}) \leq H_2(P_{\mathrm{e}}) + P_{\mathrm{e}} \log_2 \big(|\mathcal{X}| - 1\big),$$

  where $H_2(\alpha) = \alpha \log_2 \frac{1}{\alpha} + (1-\alpha) \log_2 \frac{1}{1-\alpha}$ is the binary entropy function.

- Intuition. To resolve the uncertainty in X given $\hat{X}$, we can first ask whether the two are equal, which bears uncertainty $H_2(P_{\mathrm{e}})$. In the case that they differ, which only occurs a fraction P_{e} of the time, the remaining uncertainty is at most $\log_2 \big(|\mathcal{X}| - 1\big)$, since the uniform distribution maximizes the entropy.

- Formal proof. Defining the error indicator random variable $E = \mathbb{1}\{X \neq \hat{X}\}$, we have

$$\begin{aligned}
H(X|\hat{X}) &\overset{(a)}{=} H(X, E|\hat{X}) \\
&\overset{(b)}{=} H(E|\hat{X}) + H(X|\hat{X}, E) \\
&\overset{(c)}{\leq} H(E) + H(X|\hat{X}, E) \\
&\overset{(d)}{=} H_2(P_{\mathrm{e}}) + P_{\mathrm{e}} H(X|\hat{X}, E=1) + (1 - P_{\mathrm{e}}) H(X|\hat{X}, E=0) \\
&\overset{(e)}{\leq} H_2(P_{\mathrm{e}}) + P_{\mathrm{e}} \log_2 \big(|\mathcal{X}| - 1\big),
\end{aligned}$$

  where:

  - (a) holds since E is a deterministic function of $(X, \hat{X})$. More formally, the chain rule gives $H(X, E|\hat{X}) = H(X|\hat{X}) + H(E|X, \hat{X})$, and then we have $H(E|X, \hat{X}) = 0$.
  - (b) follows from the chain rule.

- (c) holds since conditioning reduces entropy.
- (d) uses $H(E) = H_2(P_e)$ for the first term (recall that $H_2(p)$ is defined to be the entropy of a Bernoulli(p) random variable) and the definition of conditional entropy for the second term.
- (e) follows since X has no uncertainty given $\hat{X}$ when $E = 0$, and takes one of $|\mathcal{X}| - 1$ values given $\hat{X}$ when $E = 1$.

Implication for source coding.

- **Theorem.** In the block source coding problem with a discrete memoryless source P_X, if $R < H(X)$, then $P_e = \mathbb{P}[\hat{\mathbf{X}} \neq \mathbf{X}]$ cannot be made arbitrarily small as $n \to \infty$.
  - Holds for any code design! Results of this type are called *converse bounds* or *impossibility results*. (The entropy bound of the previous lecture was also of this type).
  - This is a statement of mathematical impossibility *regardless of computation, storage, etc.*
- Proof: Start with Fano's inequality with $(\mathbf{X}, \hat{\mathbf{X}})$ playing the role of the generic variables $(X, \hat{X})$:

$$H(\mathbf{X}|\hat{\mathbf{X}}) \leq H_2(P_e) + P_e \log_2 \left(|\mathcal{X}^n| - 1\right)$$

  where $\mathcal{X}^n = \mathcal{X} \times \ldots \times \mathcal{X}$ (n times) is the set of all length-n sequences with symbols in $\mathcal{X}$. For convenience, we upper bound $\log_2 \left(|\mathcal{X}^n| - 1\right) \leq \log_2 |\mathcal{X}^n| = n \log_2 |\mathcal{X}|$, and also $H_2(P_e) \leq 1$ (binary entropy is at most one bit), to obtain

$$H(\mathbf{X}|\hat{\mathbf{X}}) \leq P_e \cdot n \log_2 |\mathcal{X}| + 1$$

  This is a weakened form of Fano's inequality.

  Recall the definition of mutual information, $I(\mathbf{X}; \hat{\mathbf{X}}) = H(\mathbf{X}) - H(\mathbf{X}|\hat{\mathbf{X}})$. Upper bounding $H(\mathbf{X}|\hat{\mathbf{X}})$ according to the previous display equation gives

$$I(\mathbf{X}; \hat{\mathbf{X}}) \geq H(\mathbf{X}) - P_e \cdot n \log_2 |\mathcal{X}| - 1.$$

  On the other hand, the definition of mutual information (in the "other" form) gives

$$\begin{aligned} I(\mathbf{X}; \hat{\mathbf{X}}) &= H(\hat{\mathbf{X}}) - H(\hat{\mathbf{X}}|\mathbf{X}) \\ &\overset{(a)}{\leq} H(\hat{\mathbf{X}}) \\ &\overset{(b)}{\leq} nR \end{aligned}$$

  where (a) uses the non-negativity of (conditional) entropy, and (b) uses the fact that $\hat{\mathbf{X}}$ takes on one of $M = 2^{nR}$ values (and entropy is always upper bounded by the log of

the number of values). Combining the previous two equations with $H(\mathbf{X}) = nH(X)$ (easily verified by the i.i.d. assumption on $\mathbf{X}$, i.e., the memoryless property), we get

$$nR \geq nH(X) - P_{\mathrm{e}} \cdot n \log_2 |\mathcal{X}| - 1,$$

or equivalently,

$$P_{\mathrm{e}} \geq \frac{1}{\log_2 |\mathcal{X}|} \Big(H(X) - R - \frac{1}{n} \Big).$$

Therefore, if $R < H(X)$ then P_{e} cannot tend to zero as $n \to \infty$.

- **A minor technical detail:** On Page 44, we stated the source coding theorem for arbitrary n, not only $n \to \infty$. However, the result for $n \to \infty$ implies the result for arbitrary n. Indeed, the only way to get an arbitrarily small error probability at finite n is to have $P_{\mathrm{e}} = 0$. But if we can achieve $P_{\mathrm{e}} = 0$ at some rate with finite block length, we can also achieve it as $n \to \infty$ by simply using that code many times in succession.
- **Note:** There exist alternative proofs that show that in fact $P_{\mathrm{e}} \to 1$ as $n \to \infty$ for any source coding scheme when $R < H(X)$. That is, not only are we unable to attain a small error probability like 0.01, we can't even attain a target error probability like 0.99.

Lecture 6: Channel Coding

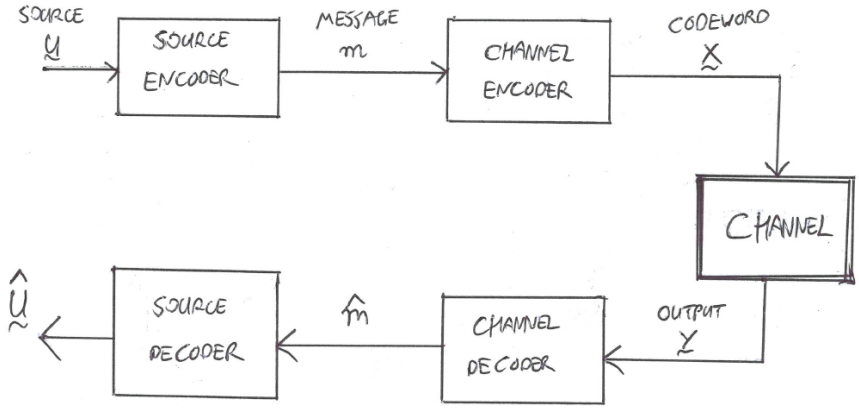
~~**Useful references:**~~

- Cover/Thomas Chapter 7
- MacKay Chapters 8–10
- Shannon's 1948 paper "A Mathematical Introduction to Communication"

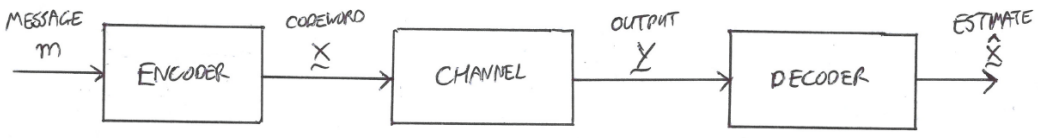
20 Setup

Overview.

- Full communication setup (source and channel coding):

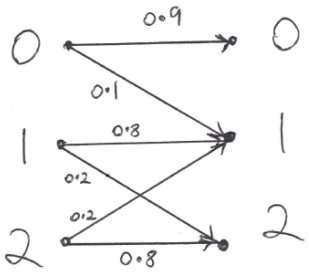


- Channel coding setup:



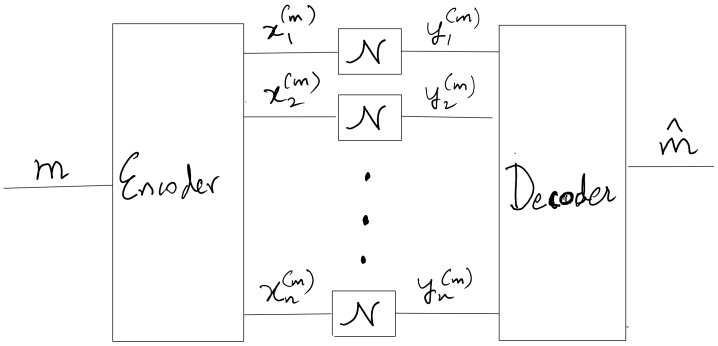
Channel model.

- The *channel* is the medium over which we transmit information.
- We denote the input by x and the output by y (or X and Y when we want to highlight that they are random).
- We assume (for now) that the channel input and output only take finitely many possible values (e.g., binary, $x \in \{0, 1\}$ and $y \in \{0, 1\}$). These sets of possible inputs/outputs are denoted by $\mathcal{X}$ and $\mathcal{Y}$. We call these the *input alphabet* and *output alphabet*.
- We adopt a *probabilistic modeling* approach: When the input is $x \in \mathcal{X}$, a given output $y \in \mathcal{Y}$ is produced with probability $P_{Y|X}(y|x)$.
- The channel transition probabilities are typically depicted graphically. A simple example:



Problem description.

- We generically view the communication problem as seeking to transmit a message $m \in \{1, \ldots, M\}$. In particular, if a fixed-length source code outputs a length-k sequence of bits, then we can set $M = 2^k$ and map each such sequence to a unique index m.



- The *encoder* takes as input the message m, and outputs a sequence of channel inputs $x_1, \ldots, x_n$. To make the dependence on the message explicit, we define the *codeword* $\mathbf{x}^{(m)} = (x_1^{(m)}, \ldots, x_n^{(m)})$, which is the sequence produced when the message is m.

- The collection of codewords $\mathcal{C} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)}\}$ is referred to as the *codebook*. It is known at both the encoder and decoder, but only the encoder knows m.

The codeword $\mathbf{x}^{(m)}$ is transmitted over the channel in n uses, and the resulting output sequence is denoted by $\mathbf{y} = (y_1, \ldots, y_n)$.

- We focus (for now) on *discrete memoryless channels*:
  - Discrete: The input/output alphabets $\mathcal{X}$ and $\mathcal{Y}$ are finite, as stated above;
  - Memoryless: When we transmit several symbols (say, n of them) over the channel in successive uses, the outputs are (conditionally) independent:

$$P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} P_{Y|X}(y_i|x_i).$$

- Given the output sequence $\mathbf{y}$ (and knowledge of the codebook $\mathcal{C}$), the *decoder* forms an estimate $\hat{m}$ of the message m.

A fundamental trade-off.

- Clearly we would like $\hat{m} = m$; if not then an *error* has occurred. Accordingly, we define the *error probability*

$$P_{\mathrm{e}} = \mathbb{P}[\hat{m} \neq m]. \tag{7}$$

We will henceforth consider this probability as being averaged over m uniform on $\{1, \ldots, M\}$ (along with the randomness in the channel), though without much extra effort we can actually get similar results for the *maximal* error probability $\max_{m=1,\ldots,M} \mathbb{P}[\hat{m} \neq m \,|\, m \text{ chosen}]$.

- We would like to transmit as much data as possible (i.e., high M); instead of considering M directly, we usually measure this via the *rate* (measured in bits per channel use):

$$R = \frac{1}{n} \log_2 M.$$

That is, the number of messages is $M = 2^{nR}$.

  - For instance, if $M = 2^n$ then $R = 1$, which makes sense because n bits (each a 0 or 1) corresponds to 2^n possible combinations (of 0s and 1s).

- The quantity n also plays a fundamental role; it is referred to as the *block length*.

- Key question: What is the fundamental trade-off between error probability P_{e}, rate R, and block length n? In particular, how high can the rate be while keeping the error probability small?

21 Channel Capacity

Definition.

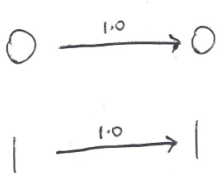
- **Definition.** The channel capacity C is defined to be the maximum[7] of all rates R such that, for any target error probability $\epsilon > 0$, there exists a block length n and codebook $\mathcal{C} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)}\}$ with $M = 2^{nR}$ codewords such that $P_{\mathrm{e}} \leq \epsilon$.
  - In simpler terms: This is the highest rate such that the error probability can be made arbitrarily small at *some* (possibly large) block length.
- **Channel Coding Theorem.** The capacity of a discrete memoryless channel $P_{Y|X}$ is

$$C = \max_{P_X} I(X;Y).$$

  The proof is split into two parts (given in later sections):
  - Achievability part: For any $R < C$, there exists a code of rate at least R with arbitrarily small error probability.
  - Converse part: For any $R > C$, any code of rate at least R cannot have arbitrarily small error probability.
- **Definition.** For a given channel $P_{Y|X}$, any input distribution P_X maximizing the mutual information above is called a *capacity-achieving input distribution.*

Examples.

- Noiseless channel:
  - Consider a noiseless channel with $\mathcal{X} = \mathcal{Y} = \{0, 1\}$ in which the output deterministically equals the input (i.e., $Y = X$):
  - An illustration:



  - Since $Y = X$, we have $H(X|Y) = 0$ (there is no uncertainty in X once we know Y), and hence

$$I(X;Y) = H(X) - H(X|Y) = H(X).$$

    Therefore, the capacity is

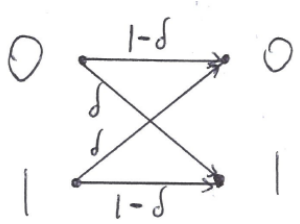
$$C = \max_{P_X} I(X;Y) = \max_{P_X} H(X) = 1$$

[7] More mathematically precisely, the supremum.

since the entropy of a binary random variable is at most one (achieved when $P_X(0) = P_X(1) = \frac{1}{2}$).

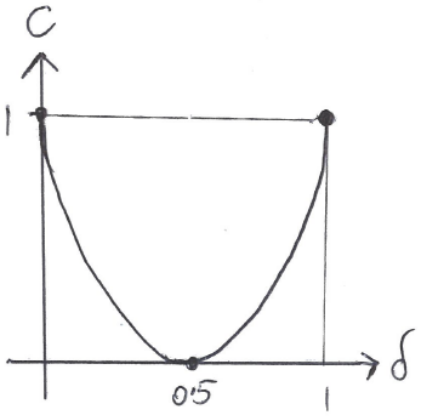
- This result should not be surprising – if there is no noise, we can reliably transmit one bit per channel use without even doing any coding!

- Binary symmetric channel:
  - Again consider $\mathcal{X} = \mathcal{Y} = \{0, 1\}$, but now each input is flipped with some probability $\delta \in (0, 1)$:
$$P_{Y|X}(y|x) = \begin{cases} 1-\delta & y = x \\ \delta & y = 1-x. \end{cases}$$
  - An illustration:



  - In this case, it is more convenient to use the expansion $I(X;Y) = H(Y) - H(Y|X)$.
  - In general we have $H(Y|X) = \sum_x P_X(x) H(Y|X = x)$, but due to the symmetry things simplify. Specifically, regardless of whether we condition on $X = 0$ or $X = 1$, the conditional probabilities of Y are still δ and $1-\delta$, and so $H(Y|X = x) = H_2(\delta)$, where $H_2(\delta) = \delta \log_2 \frac{1}{\delta} + (1-\delta) \log_2 \frac{1}{1-\delta}$ is the binary entropy function.
  - This gives $H(Y|X) = H_2(\delta)$ and hence
$$C = \max_{P_X} I(X;Y) = \max_{P_X} \big(H(Y) - H_2(\delta)\big).$$
    If we were maximizing over P_Y directly, we could get $\max H(Y) = 1$ by the same argument as the noiseless case by letting P_Y be uniform. But in this case, even though we can only control P_X, we can still produce uniform P_Y – just let P_X be uniform!
    - Indeed, if $P_X(0) = P_X(1) = \frac{1}{2}$, then we have
$$P_Y(0) = \frac{1}{2}(1-\delta) + \frac{1}{2}\delta = \frac{1}{2},$$
      and similarly $P_Y(1) = \frac{1}{2}$.
  - Therefore, the capacity is
$$C = 1 - H_2(\delta)$$
    and the capacity-achieving input distribution is $P_X(0) = P_X(1) = \frac{1}{2}$.

- An illustration:



- As expected, setting $\delta = 0$ recovers the noiseless capacity $C = 1$. Notice also that $\delta = \frac{1}{2}$ gives capacity zero, because in this case we have $P_{Y|X}(y|x) = \frac{1}{2}$ regardless of the input x, so the output carries no information about the input.

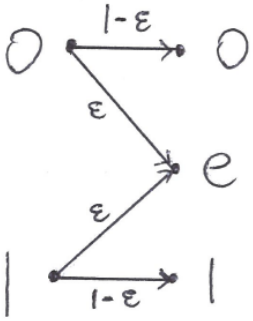
- Binary erasure channel:

  - Consider $\mathcal{X} = \{0, 1\}$, $\mathcal{Y} = \{0, 1, e\}$, and transition probabilities

$$P_{Y|X}(y|x) = \begin{cases} 1-\epsilon & y = x \\ \epsilon & y = e \\ 0 & y = 1-x \end{cases}$$

    for some *erasure probability* ϵ. In words, the output equals the input with probability $1 - \epsilon$, but is "erased" (corresponding to output e) with probability ϵ.

  - An illustration:



  - This time it turns out easier to use the expansion $I(X;Y) = H(X) - H(X|Y)$, though the $I(X;Y) = H(Y) - H(Y|X)$ approach is also possible (see the tutorial).
  - $H(X|Y)$ is fairly easy to characterize, because $H(X|Y = 0) = H(X|Y = 1) = 0$ (there is no uncertainty in X when $Y \neq e$). Hence,

$$H(X|Y) = \sum_y P_Y(y) H(X|Y = y) = P_Y(e) H(X|Y = e).$$

Then, given $Y = e$, we have

$$P_{X|Y}(0|e) = \frac{P_{XY}(0,e)}{P_Y(e)} = \frac{P_X(0)\epsilon}{\epsilon} = P_X(0),$$

and similarly $P_{X|Y}(1|e) = P_X(1)$. Hence, $H(X|Y = e) = H(X)$.

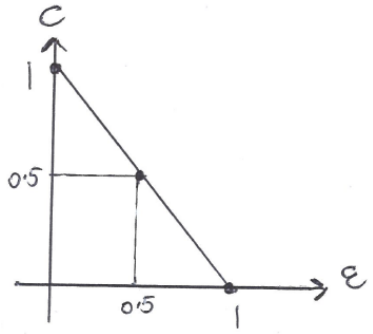
- Combining the above findings gives

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) \\ &= (1-\epsilon)H(X). \end{aligned}$$

- Upon maximizing over P_X, we can get the maximal value $H(X) = 1$ with $P_X(0) = P_X(1) = \frac{1}{2}$. Therefore, the capacity is

$$C = 1 - \epsilon.$$

An illustration:



- In all of these examples, the capacity-achieving input distribution is uniform.
  - In fact, much more general classes of symmetric channels (not necessarily binary) have a uniform capacity-achieving input distribution. See Cover/Thomas Section 7.2 for details.
  - For non-symmetric channels, the capacity-achieving P_X may be non-uniform. Moreover, we often can't find the optimal choice analytically, so instead, we need to do so numerically (efficient algorithms for doing this are known; see Cover/Thomas Section 10.8).

22 Jointly Typical Sequences

The following definition and properties will be crucial in proving the achievability part mentioned above.

- **Definition**: A pair $(\mathbf{x}, \mathbf{y})$ of length-n input and output sequences is said to be *jointly typical* with respect to a joint distribution P_{XY} if the following conditions hold:

$$\begin{aligned} 2^{-n(H(X)+\epsilon)} &\leq P_{\mathbf{X}}(\mathbf{x}) \leq 2^{-n(H(X)-\epsilon)} \\ 2^{-n(H(Y)+\epsilon)} &\leq P_{\mathbf{Y}}(\mathbf{y}) \leq 2^{-n(H(Y)-\epsilon)} \\ 2^{-n(H(X,Y)+\epsilon)} &\leq P_{\mathbf{XY}}(\mathbf{x}, \mathbf{y}) \leq 2^{-n(H(X,Y)-\epsilon)}. \end{aligned}$$

  The set of all such sequences is denoted by $\mathcal{T}_n(\epsilon)$, and is called the *jointly typical set.*

  - In simpler terms: The X sequence, Y sequence, and joint (X, Y) sequence are all typical according to the previous lecture's definition.

- **Key properties**:[8]

  1. (Equivalent definition) We have $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}_n(\epsilon)$ if and only if the following conditions hold:

$$\begin{aligned} H(X) - \epsilon \leq \frac{1}{n} \sum_{i=1}^{n} \log_2 \frac{1}{P_X(x_i)} \leq H(X) + \epsilon \\ H(Y) - \epsilon \leq \frac{1}{n} \sum_{i=1}^{n} \log_2 \frac{1}{P_Y(y_i)} \leq H(Y) + \epsilon \\ H(X,Y) - \epsilon \leq \frac{1}{n} \sum_{i=1}^{n} \log_2 \frac{1}{P_{XY}(x_i, y_i)} \leq H(X,Y) + \epsilon. \end{aligned}$$

  2. (High probability) $\mathbb{P}[(\mathbf{X}, \mathbf{Y}) \in \mathcal{T}_n(\epsilon)] \to 1$ as $n \to \infty$.
  3. (Cardinality upper bound) $|\mathcal{T}_n(\epsilon)| \leq 2^{n(H(X,Y)+\epsilon)}$.
  4. (Probability for independent sequences) If $(\mathbf{X}', \mathbf{Y}') \sim P_{\mathbf{X}}(\mathbf{x}')P_{\mathbf{Y}}(\mathbf{y}')$ are independent copies of $(\mathbf{X}, \mathbf{Y})$, then the probability of joint typicality is

$$\mathbb{P}[(\mathbf{X}', \mathbf{Y}') \in \mathcal{T}_n(\epsilon)] \leq 2^{-n(I(X;Y)-3\epsilon)}.$$

  - The first three properties have similar intuition to the "X-only" setting of the previous lecture.
  - The final one is distinct from that setting. Intuitively, if $\mathbf{X}'$ and $\mathbf{Y}'$ are generated independently, then the "further" P_{XY} is from being independent, the less likely it is for those independent sequences to be jointly typical with respect to P_{XY}. Mutual information naturally arises because it measures "how far" (X, Y) are from being independent: $I(X;Y) = D(P_{XY} \| P_X \times P_Y)$.
  - In fact, the fourth property is a special case of a more general result: If a sequence $\mathbf{Z} = (Z_1, \ldots, Z_n)$ is drawn i.i.d. from some distribution Q_Z, then the probability that it is typical with respect to some other distribution P_Z is roughly $2^{-nD(P_Z \| Q_Z)}$.

[8] Near-matching lower bounds can also be shown for the final two properties, but these are omitted here.

- **Proofs**:

  1. Simple re-arranging like in the previous lecture.
  2. Law of large numbers applied (separately) to the 3 conditions in the first property.
  3. Same as the previous lecture via $P_{\mathbf{XY}}(\mathbf{x},\mathbf{y}) \geq 2^{-n(H(X,Y)+\epsilon)}$ and $\sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}_n(\epsilon)} P_{\mathbf{X},\mathbf{Y}}(\mathbf{x},\mathbf{y}) \leq 1$.
  4. We have

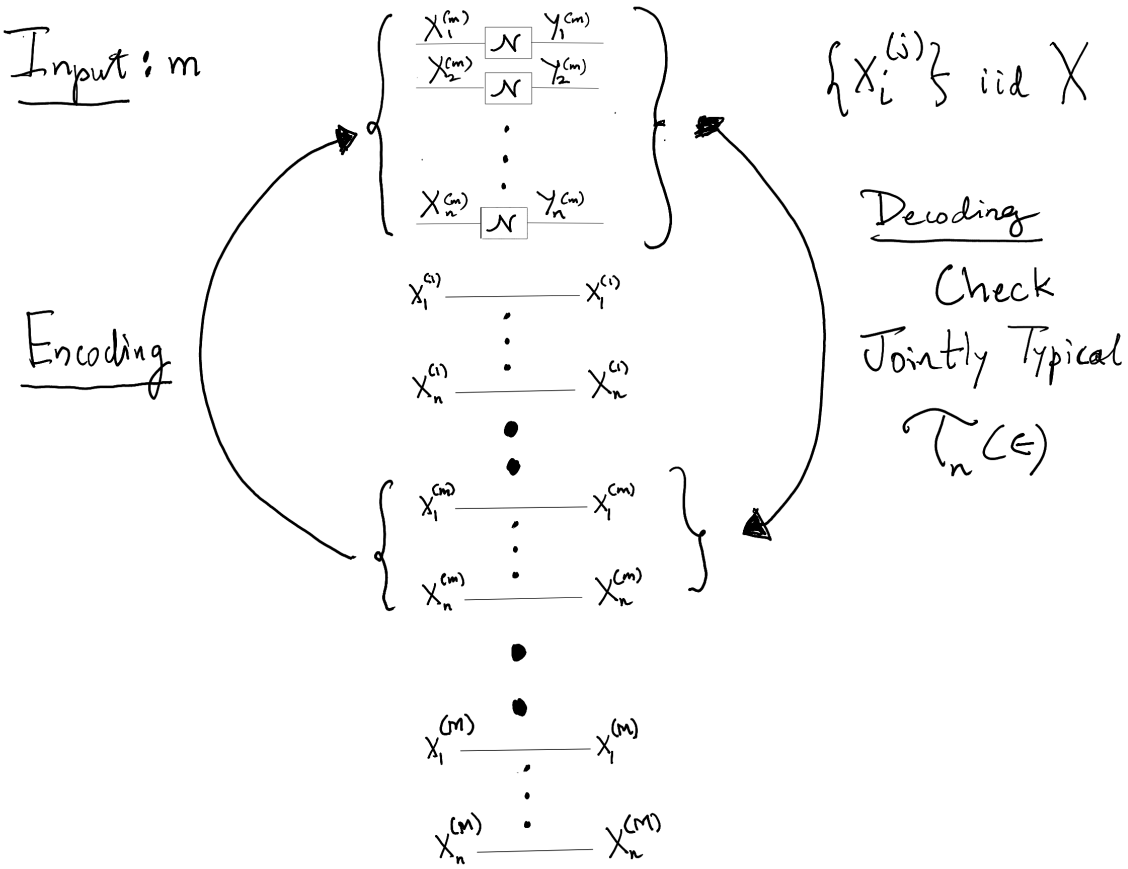
$$\begin{aligned}
\mathbb{P}[(\mathbf{X}',\mathbf{Y}') \in \mathcal{T}_n(\epsilon)] &= \sum_{(\mathbf{x}',\mathbf{y}')\in\mathcal{T}_n(\epsilon)} P_{\mathbf{X}}(\mathbf{x}')P_{\mathbf{Y}}(\mathbf{y}') \\
&\overset{(a)}{\leq} \sum_{(\mathbf{x}',\mathbf{y}')\in\mathcal{T}_n(\epsilon)} 2^{-n(H(X)-\epsilon)}2^{-n(H(Y)-\epsilon)} \\
&\overset{(b)}{\leq} 2^{n(H(X,Y)+\epsilon)}2^{-n(H(X)-\epsilon)}2^{-n(H(Y)-\epsilon)} \\
&\overset{(c)}{=} 2^{-n(I(X;Y)-3\epsilon)},
\end{aligned}$$

  where (a) uses the fact that $P_{\mathbf{X}}(\mathbf{x}') \leq 2^{-n(H(X)-\epsilon)}$ and $P_{\mathbf{Y}}(\mathbf{y}') \leq 2^{-n(H(Y)-\epsilon)}$ within $\mathcal{T}_n(\epsilon)$, (b) uses the upper bound in property 3, and (c) uses $I(X;Y) = H(X) + H(Y) - H(X,Y)$.

23 Achievability via Random Coding

Overview.

- Challenge: Devising explicit/specific codes and studying their performance is very difficult.
- Key idea (the probabilistic method): Show that **randomly chosen** codes perform well on average. Obviously, the best possible code must perform at least as well as the average.
- Note: The good code whose existence we prove may have very high computation/storage requirements. This approach merely shows that reliable communication is *mathematically* possible for rates below capacity but not how to get there with a *practical/efficient* design.



Codebook generation.

- Recall that the encoding is done via a codebook $\mathcal{C} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)}\}$, where message m is encoded into the length-n sequence $\mathbf{x}^{(m)} = (x_1^{(m)}, \ldots, x_n^{(m)})$.
- We consider the following *random coding* approach:

  Generate each symbol $X_i^{(m)}$ of each codeword randomly and independently according to some distribution P_X (to be specified)

  Note that we use capital letters for $\mathbf{X}$, $X_i^{(m)}$, etc., when we want to highlight that they are random.

- For example, if $\mathcal{X} = \{0, 1\}$ and $P_X(1) = P_X(0) = \frac{1}{2}$, then we are just setting every bit of every codeword according to a fair "coin flip".

Encoding and decoding.

- As mentioned above, the encoder simply maps m to $\mathbf{X}^{(m)} = (X_1^{(m)}, \ldots, X_n^{(m)}) \in \mathcal{X}^n$, which is transmitted via n uses of the channel.

- The decoder receives the output sequence $\mathbf{Y} = (Y_1, \ldots, Y_n)$, and also knows the codebook. For each $\tilde{m} = 1, \ldots, M$, it checks whether the pair $(\mathbf{X}^{(\tilde{m})}, \mathbf{Y})$ is jointly typical, and does the following:
  - If there exists a unique $\tilde{m}$ that joint typicality holds, then the decoder estimates $\hat{m} = \tilde{m}$.
  - If there exists no such $\tilde{m}$, or multiple such $\tilde{m}$, an error is declared (or alternatively, $\hat{m}$ is simply chosen at random).

  Note that "joint typicality" is defined with respect to $P_{XY} = P_X \times P_{Y|X}$. The channel $P_{Y|X}$ was fixed as part of the problem, whereas P_X is something we chose ourselves (during the codebook generation).

- Note that the joint distributions between the codewords and the output are exactly those we need to apply properties 2 and 4 of joint typicality:
  - For the correct m (i.e., $\mathbf{X}^{(m)}$ is transmitted), $P_{\mathbf{Y}|\mathbf{X}}$ is i.i.d. according to $P_{Y|X}$, and $\mathbf{X}^{(m)}$ itself is i.i.d. according to P_X by construction, so overall $(\mathbf{X}^{(m)}, \mathbf{Y})$ is i.i.d. on $P_{XY} = P_X \times P_{Y|X}$.
  - For any incorrect $\tilde{m}$ (i.e., $\mathbf{X}^{(\tilde{m})}$ is a non-transmitted codeword), we have that $\mathbf{X}^{(\tilde{m})}$ and $\mathbf{Y}$ are independent, since $\mathbf{Y}$ only depends on the transmitted codeword, not the other ones. Therefore, the joint distribution of $(\mathbf{X}^{(\tilde{m})}, \mathbf{Y})$ takes the form $P_{\mathbf{X}}(\mathbf{x})P_{\mathbf{Y}}(\mathbf{y})$.

Analysis of the error probability.

- In order to have $\hat{m} = m$, it is clearly sufficient that the following two events occur:
  1. $(\mathbf{X}^{(m)}, \mathbf{Y})$ is jointly typical;
  2. None of the other $(\mathbf{X}^{(\tilde{m})}, \mathbf{Y})$ are jointly typical (with $\tilde{m} \neq m$).

- Let $\overline{P}_{\mathrm{e}}^{(m)}$ denote the error probability given that the message is m, averaged over both the randomness in the channel *and* the random codebook (previously we only averaged over the former). This is called the *random-coding error probability*.

- We have just argued that the success probability $1 - \overline{P}_{\mathrm{e}}^{(m)}$ satisfies

$$1 - \overline{P}_{\mathrm{e}}^{(m)} \geq \mathbb{P}\left[(\mathbf{X}^{(m)}, \mathbf{Y}) \in \mathcal{T}_n(\epsilon) \cap \bigcap_{\tilde{m} \neq m} \left\{(\mathbf{X}^{(\tilde{m})}, \mathbf{Y}) \notin \mathcal{T}_n(\epsilon)\right\}\right],$$

  which, by de Morgan's laws, is equivalent to

$$\overline{P}_{\mathrm{e}}^{(m)} \leq \mathbb{P}\left[(\mathbf{X}^{(m)}, \mathbf{Y}) \notin \mathcal{T}_n(\epsilon) \cup \bigcup_{\tilde{m} \neq m} \left\{(\mathbf{X}^{(\tilde{m})}, \mathbf{Y}) \in \mathcal{T}_n(\epsilon)\right\}\right].$$

- Using the union bound $\mathbb{P}[A_i \cup \ldots \cup A_N] \leq \sum_{i=1}^{N} \mathbb{P}[A_i]$, we obtain

$$\overline{P}_{\mathrm{e}}^{(m)} \leq \mathbb{P}\big[(\mathbf{X}^{(m)}, \mathbf{Y}) \notin \mathcal{T}_n(\epsilon)\big] + \sum_{\tilde{m} \neq m} \mathbb{P}\big[(\mathbf{X}^{(\tilde{m})}, \mathbf{Y}) \in \mathcal{T}_n(\epsilon)\big].$$

- By the i.i.d. random coding method and the memoryless property of the channel, $(\mathbf{X}^{(m)}, \mathbf{Y})$ is i.i.d. on P_{XY}. Moreover, since $\mathbf{X}^{(m)}$ is the only codeword that $\mathbf{Y}$ depends on, we also have that $(\mathbf{X}^{(\tilde{m})}, \mathbf{Y})$ is an independent pair with the same $P_{\mathbf{X}}$ and $P_{\mathbf{Y}}$ marginals as $(\mathbf{X}^{(m)}, \mathbf{Y})$.

- Therefore, the joint typicality properties in the previous section give $(\mathbf{X}^{(m)}, \mathbf{Y}) \in \mathcal{T}_n(\epsilon)$ with probability approaching one (as n increases), and that the probability of $(\mathbf{X}^{(\tilde{m})}, \mathbf{Y}) \in \mathcal{T}_n(\epsilon)$ is at most $2^{-n(I(X;Y)-3\epsilon)}$, which gives

$$\begin{aligned}
\overline{P}_{\mathrm{e}}^{(m)} &\leq \mathbb{P}\big[(\mathbf{X}^{(m)}, \mathbf{Y}) \notin \mathcal{T}_n(\epsilon)\big] + \sum_{\tilde{m} \neq m} \mathbb{P}\big[(\mathbf{X}^{(\tilde{m})}, \mathbf{Y}) \in \mathcal{T}_n(\epsilon)\big] \\
&\overset{(a)}{\leq} \delta_n + \sum_{\tilde{m} \neq m} 2^{-n(I(X;Y)-3\epsilon)} \\
&\overset{(b)}{\leq} \delta_n + M \times 2^{-n(I(X;Y)-3\epsilon)}, \\
&\leq \ \delta_n + 2^{-n(I(X;Y)-3\epsilon-R)}.
\end{aligned}$$

  where in (a) δ_n denotes a sequence that tends to 0 as $n \to \infty$, and in (b) we used the fact that the number of terms in the summation is $M - 1 \leq M$.

- Since $M = 2^{nR}$, we find that for $R < I(X;Y) - 3\epsilon$ the overall upper bound on $\overline{P}_{\mathrm{e}}^{(m)}$ tends to zero as $n \to \infty$. Since ϵ may be arbitrarily small, this means $\overline{P}_{\mathrm{e}}^{(m)}$ can be made arbitrarily small for any rate R arbitrarily close to $I(X;Y)$.

- Since this holds for any m, it also holds for the random-coding error probability $\frac{1}{M}\sum_{m=1}^{M} \overline{P}_{\mathrm{e}}^{(m)}$ averaged over the message m. (In fact, due to the symmetry of random coding, $\overline{P}_{\mathrm{e}}^{(m)}$ is the same for all m.)

- Finally, by choosing P_X to achieve the maximum in the definition $C = \max_{P_X} I(X;Y)$, we deduce that we can get vanishing error probability for rates arbitrarily close to the capacity C.

(Optional) Alternative proof.

- In an interesting alternative proof, instead of the notion of joint typicality we considered in the discrete setting, the decoder looks for a codeword $\mathbf{x}$ such that

$$\sum_{i=1}^{n} \log_2 \frac{P_{Y|X}(y_i|x_i)}{P_Y(y_i)} \geq \gamma$$

  for some threshold γ. This can be viewed as a form of *one-sided* typicality.

- Using a simple change of measure argument, one can show that a given *incorrect* codeword passes this threshold test with probability at most $2^{-\gamma}$. By the union bound, the probability of this occurring for *any* incorrect codeword is at most $M2^{-\gamma}$, which tends to zero if we set γ to be slightly above $\log_2 M$.

- By the law of large numbers, for the *correct* codeword, $\sum_{i=1}^{n} \log_2 \frac{P_{Y|X}(y_i|x_i)}{P_Y(y_i)}$ is close to $nI(X;Y)$ with high probability. Therefore, to exceed the threshold $\gamma \approx \log_2 M = nR$, we just need $R < I(X;Y)$.

- This proof is rooted in two early works: "Certain results in coding theory for noisy channels" (Shannon, 1957) and "A new basic theorem of information theory" (Feinstein, 1954).

24 Converse via Fano's Inequality

- Let $\mathbf{m}$ denote a transmitted message uniform on $\{1, \ldots, M\}$, and let $\hat{\mathbf{m}}$ be its estimate (in a slight shift from our usual convention, these are random variables even though they are written in lower-case).

- The error probability is $P_{\mathrm{e}} = \mathbb{P}[\hat{\mathbf{m}} \neq \mathbf{m}]$. Fano's inequality from the previous lecture[9] states that

$$\begin{aligned} H(\mathbf{m}|\hat{\mathbf{m}}) &\leq H_2(P_{\mathrm{e}}) + P_{\mathrm{e}} \log_2(M-1) \\ &\leq 1 + P_{\mathrm{e}} \log_2 M. \end{aligned}$$

- Since $\mathbf{m}$ is uniform on $\{1, \ldots, M\}$, we have $H(\mathbf{m}) = \log_{\mathbf{2}} \mathbf{M}$, which gives

$$\begin{aligned} I(\mathbf{m};\hat{\mathbf{m}}) &= H(\mathbf{m}) - H(\mathbf{m}|\hat{\mathbf{m}}) \\ &\geq \log_2 M - P_{\mathrm{e}} \log_2 M - 1 \\ &= (1 - P_{\mathrm{e}}) \log_2 M - 1, \end{aligned}$$

  where the inequality uses the previous display equation. Simple re-arranging gives

$$P_{\mathrm{e}} \geq 1 - \frac{I(\mathbf{m};\hat{\mathbf{m}}) + \mathbf{1}}{\log_2 M}.$$

  Intuitively, this says that to achieve a small error probability, we need the amount of information that $\hat{\mathbf{m}}$ reveals about $\mathbf{m}$ to be close to the prior uncertainty in $\mathbf{m}$ (which is $\log_2 M$).

[9] Now with $(\mathbf{m}, \hat{\mathbf{m}})$ in place of the generic symbols $(X, \hat{X})$ used in that lecture.

- The key step is to bound the mutual information. We have:

$$\begin{aligned} I(\mathbf{m};\hat{\mathbf{m}}) &\overset{(a)}{\leq} I(\mathbf{X};\mathbf{Y}) \\ &\overset{(b)}{=} H(\mathbf{Y}) - H(\mathbf{Y}|\mathbf{X}) \\ &\overset{(c)}{\leq} \sum_{i=1}^{n} H(Y_i) - H(\mathbf{Y}|\mathbf{X}) \\ &\overset{(d)}{=} \sum_{i=1}^{n} H(Y_i) - \sum_{i=1}^{n} H(Y_i|\mathbf{X}) \\ &\overset{(e)}{=} \sum_{i=1}^{n} H(Y_i) - \sum_{i=1}^{n} H(Y_i|X_i) \\ &\overset{(f)}{=} \sum_{i=1}^{n} I(X_i;Y_i) \\ &\overset{(g)}{\leq} nC, \end{aligned}$$

  where:

  - (a) uses the data processing inequality (note that $\mathbf{m} \to \mathbf{X} \to \mathbf{Y} \to \hat{\mathbf{m}}$ forms a Markov chain);
  - (b) and (f) use the definition of mutual information;
  - (c) uses the sub-additivity of entropy;
  - (d) uses the fact that the Y_i are conditionally independent given $\mathbf{X}$ (and entropy is additive for independent random variables), i.e., the "memoryless" assumption;
  - (e) uses the fact that Y_i depends on $\mathbf{X}$ only through X_i;
  - (g) uses the definition of capacity (C is the maximum mutual information between X and Y).

- Combining the previous two dot points with $\log_2 M = \log_2 2^{nR} = nR$ gives

$$P_{\text{e}} \geq 1 - \frac{C + 1/n}{R},$$

  which means that P_{e} is bounded away from 0 as $n \to \infty$ whenever $R > C$.

  - **A minor technical detail:** We originally stated the channel coding theorem for arbitrary n, not only $n \to \infty$. However, the result for $n \to \infty$ implies the result for arbitrary n. Indeed, the only way to get arbitrarily small error probability at finite n is to have $P_{\text{e}} = 0$. But if we can achieve $P_{\text{e}} = 0$ at some rate with finite block length, we can also achieve it as $n \to \infty$ by simply using that codebook many times in succession.

25 (Optional) Joint Source-Channel Coding

- If we can successfully perform both source coding and channel coding, then we can form the overall communication system as shown in the first figure of this document (Page 1).

- Denoting the source block length by k and the channel block length by n, and taking both to be sufficiently large, we obtain the following condition for overall reliable communication:

$$\underbrace{n \times C}_{\text{Total Capacity}} \quad > \quad \underbrace{k \times H}_{\text{Total Entropy}}$$

  or equivalently

$$\frac{k}{n} < \frac{C}{H}.$$

  Indeed, this result follows from a simple combination of the source coding and channel coding theorems. We first compress the source and represent it using roughly $M \approx 2^{kH}$ bits, and then we send the corresponding index $m \in \{1, \ldots, M\}$ across the channel in n uses.

- It may seem strange that we are removing first redundancy (source coding) only to then add redundancy (channel coding) – could a joint approach be better? This is known as *joint source-channel coding*.

- **Separation theorem.** Even with joint source-channel coding, reliable communication is impossible if $\frac{k}{n} > \frac{C}{H}$. Therefore, separate source-channel coding is asymptotically optimal at large block lengths.

  - Proof: Mostly similar to that above based on Fano's inequality. See Section 7.13 of Cover/Thomas.
  - Note: The gains can be significant at *finite* block lengths (beyond the scope of this course).

Lecture 7: Practical Channel Codes

Useful references:

- Blog post on Hamming codes[10]
- Cover/Thomas Section 7.11
- MacKay Sections 11.4–11.5, Chapters 13–14
- (Beyond the scope of this course) Survey article on coding,[11] and/or the advanced textbook "Modern Coding Theory" (Richardson and Urbanke)

26 Recap of Parity Checks and the Hamming Code

Parity check.

- A *parity check* of a sequence of bits $b_1, \ldots, b_m$ is an additional bit equaling 1 if the number of 1's in $b_1 \ldots, b_m$ is odd, and 0 if the number of 1's is even.
    - Hence, in either case, there is an even number of 1's in the sequence $b_1, \ldots, b_m c$, where c is the parity check bit
- We can express this via modulo-2 arithmetic: For $a, b \in \{0, 1\}$, let the $\oplus$ operator be defined as

$$0 \oplus 0 = 1 \oplus 1 = 0$$
$$0 \oplus 1 = 1 \oplus 0 = 1.$$

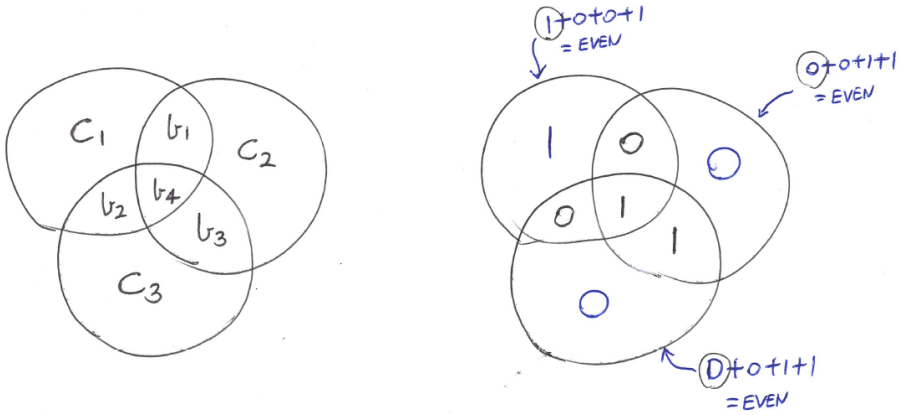
  Then the parity check of $b_1, \ldots, b_m$ is $c = b_1 \oplus \ldots \oplus b_m$.

[10] `https://jeremykun.com/2015/03/02/hammings-code/`
[11] `https://arxiv.org/pdf/1908.09903.pdf`

- If we transmit the length-$(m+1)$ sequence $b_1 \dots b_m c$ across a channel, and one of the bits is flipped, we will notice that the total number of 1's is no longer even. Hence, we can *detect* a single bit flip. However, a little thought reveals that we cannot *correct* it.
- The idea that permits error correction: *Send multiple parity checks applied to different groups of bits*

Simple examples.

- In this first lecture, we introduced the *repetition code*, which (for example) encodes 1010 into 111000111000. By a majority vote rule, this permits the *correction* of one bit flip in each of the groups of 3 bits.
- We also introduced the Hamming code, which maps 4 bits to 7 bits and can correct a single bit flip:



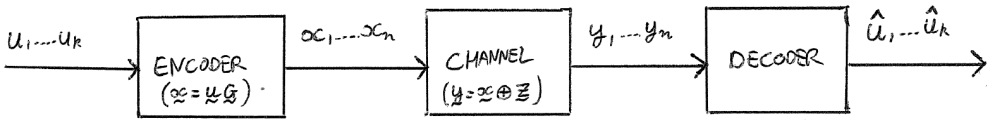
We described the check bits $c_1c_2c_3$ as being chosen to make the number of 1's per circle even. In the above terminology, each c_i is a *parity check* of the b_i's in the corresponding circle.

  - Note: When we talk about being able to correct a certain number of bit flips, this may include flips in both the uncoded bits b_i and the check bits c_i.
  - How to correct one bit flip: Observe that each possible single bit flip changes a unique combination of circles from "even number of 1s" to "odd number of 1s" (e.g., the middle bit changes all 3 circles; the top-left bit only changes the top-left circle). Therefore, the decoder can check which circles have an odd number of 1s, and un-flip the corresponding bit. If the decoder sees all 3 circles already having an even number of 1s, then no changes are made.

27 Linear Codes

Notation.

- We will now switch notation a little, and consider the general procedure of mapping k bits $\mathbf{u} = (u_1, \ldots, u_k)$ to n bits $\mathbf{x} = (x_1, \ldots, x_n)$, where $n \geq k$.
- We will mostly consider the transmission of $\mathbf{x} = (x_1, \ldots, x_n)$ across a binary symmetric channel (BSC) to produce $\mathbf{y} = (y_1, \ldots, y_n)$; these output bits are used to construct an estimate $\hat{\mathbf{u}} = (\hat{u}_1, \ldots, \hat{u}_k)$ of the original k bits.



The channel can be described as $\mathbf{y} = \mathbf{x} \oplus \mathbf{z}$, where $\mathbf{z} \in \{0,1\}^n$ indicates which bits are flipped, and the operator $\oplus$ is applied bit-by-bit.

- Notice that this is the channel coding setup of the previous lectures specialized to the BSC. The generic "message" $m \in \{1, \ldots, M\}$ is now replaced by "message bits" $(u_1, \ldots, u_k) \in \{0,1\}^k$, so that $M = 2^k$. Previously we defined the rate as $R = \frac{1}{n} \log_2 M$, and substituting $M = 2^k$ gives

$$R = \frac{k}{n}.$$

  This is intuitive: If we map k bits to $3k$ bits (say), then the rate is $\frac{1}{3}$.

Definition and generator matrix.

- For reasons to be made clear shortly, we say that any code comprised of parity checks is a *linear code*.
- We distinguish between the following two cases:
  - A **systematic parity-check code** is one in which the first k (out of n) bits of $\mathbf{x}$ are always precisely the original k bits, and the remaining $n - k$ bits are parity checks:

$$x_i = u_i, \quad i = 1, \ldots, k,$$
$$x_i = \bigoplus_{j=1}^{k} u_j g_{j,i}, \quad i = k+1, \ldots, n$$

    where $g_{j,i} = 1$ if the parity check in location i includes u_j, and $g_{j,i} = 0$ otherwise. For instance, the Hamming code described above is a systematic parity-check code.
  - A **general parity-check code** is one in which all n codeword bits may be arbitrary parity checks:

$$x_i = \bigoplus_{j=1}^{k} u_j g_{j,i}, \quad i = 1, \ldots, n.$$

    Clearly a systematic code is a special case of this, since it corresponds to setting $g_{j,i} = \mathbf{1}\{j = i\}$ for $i = 1, \ldots, k$.
- The above (general) formula for generating each x_i from $u_1, \ldots, u_k$ can be succinctly summarized as a (modulo-2) vector-matrix multiplication:

$$\mathbf{x} = \mathbf{uG}, \quad \text{(in modulo-2 arithmetic)}$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{u} = (u_1, \ldots, u_k)$ are the suitable row vectors, and

$$\mathbf{G} = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,n} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,1} & g_{k,2} & \cdots & g_{k,n} \end{bmatrix}$$

is known as the *generator matrix*.

  - Interpretation: The 1's in each column indicate which bits are included in the parity check

- In the special case of a systematic code, this simplifies to

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & \ldots & 0 & g_{1,k+1} & \cdots & g_{1,n} \\ 0 & 1 & \ldots & 0 & g_{2,k+1} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 & g_{k,k+1} & \cdots & g_{k,n} \end{bmatrix}$$

  with the left-most $k \times k$ sub-matrix being the identity matrix.

- With the mapping from $\mathbf{u}$ to $\mathbf{x}$ being described by the matrix multiplication $\mathbf{x} = \mathbf{uG}$ (in modulo-2 arithmetic), we can now justify the terminology *linear code*: If $\mathbf{u}$ and $\mathbf{u}'$ are two different message sequences, and their corresponding codewords are $\mathbf{x} = \mathbf{uG}$ and $\mathbf{x}' = \mathbf{uG}'$, then

$$\begin{aligned} \mathbf{x} \oplus \mathbf{x}' &= \mathbf{uG} \oplus \mathbf{u}'\mathbf{G} \\ &= (\mathbf{u} \oplus \mathbf{u}')\mathbf{G}, \end{aligned}$$

  which means that $\mathbf{x} \oplus \mathbf{x}'$ is also a codeword (corresponding to message $\mathbf{u} \oplus \mathbf{u}'$). In other words, the (modulo-2) sum of any two valid codewords is another valid codeword.

  - Note: We have extended the $\oplus$ notation to vectors/sequences, which is done via a bit-by-bit application of the definition above, e.g., $0001 \oplus 1101 = 1100$ and $101 \oplus 010 = 111$.

- **Examples.** With $k = 4$, the generator matrices for the single-parity-check code and Hamming code (described at the start of the lecture) are given by

$$\mathbf{G}_{\text{parity}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \qquad \mathbf{G}_{\text{Hamming}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{8}$$

  Sometimes you might see the latter with the last 3 columns in a different order, but re-ordering columns just amounts to re-labeling the indices of parity check bits.

- Pre-multiplying $\mathbf{G}_{\text{Hamming}}$ by all 16 possible $\mathbf{u}$ sequences, we can list all the codewords of the Hamming code:

$$\begin{array}{cccc} 0000000 & 0001111 & 0010011 & 0011100 \\ 0100101 & 0101010 & 0110110 & 0111001 \\ 1000110 & 1001001 & 1010101 & 1011010 \\ 1100011 & 1101100 & 1110000 & 1111111. \end{array}$$

- For instance, the codeword 1100011 is obtained from $\mathbf{u} = 1100$ by taking the modulo-2 sum of the first two rows of $\mathbf{G}_{\text{Hamming}}$. For $\mathbf{u} = 1111$, we take the modulo-2 sum of all 4 rows.

- As we will see in the tutorial, any general code can be reduced to an "equivalent" systematic code by applying a technique similar to Gaussian elimination on $\mathbf{G}$.

Parity-check matrix.

- A matrix closely related to $\mathbf{G}$, but which will be more directly useful when it comes to decoding, is called the *parity-check matrix* $\mathbf{H}$. It is an $n \times (n-k)$ matrix that satisfies

$$\mathbf{xH} = \mathbf{0} \iff \mathbf{x} \text{ is a valid codeword.}$$

  Notice the distinction:

  - $\mathbf{G}$ is used to *generate* $\mathbf{x}$ from $\mathbf{u}$;
  - $\mathbf{H}$ is used to *check* if $\mathbf{x}$ can be generated from *any* $\mathbf{u}$ (doing this naively using $\mathbf{G}$ by testing all 2^k possible $\mathbf{u}$ would be grossly inefficient).

- While such check matrices exist for all generator matrices, we will focus our attention on the systematic case, as it is much simpler. Recall the two formulas we gave for x_i (in terms of the $\{u_j\}$ and $\{g_{j,i}\}$) in the systematic case; substituting the first into the second gives

$$x_i = \bigoplus_{j=1}^{k} x_j g_{j,i}, \quad i = k+1, \ldots, n.$$

  Adding x_i (modulo 2) to both sides, the left-hand side gives $x_i \oplus x_i = 0$, and we are left with

$$\left(\bigoplus_{j=1}^{k} x_j g_{j,i} \right) \oplus x_i = 0, \quad i = k+1, \ldots, n.$$

Converting to matrix form reveals that the following choice of $\mathbf{H}$ indeed gives $\mathbf{xH} = \mathbf{0}$:

$$\mathbf{H} = \begin{bmatrix} g_{1,k+1} & g_{1,k+2} & \cdots & g_{1,n} \\ g_{2,k+1} & g_{2,k+2} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,k+1} & g_{k,k+2} & \cdots & g_{k,n} \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}.$$

It is also not hard to establish the opposite, i.e., if $\mathbf{xH} = \mathbf{0}$ then $\mathbf{x}$ is indeed a valid codeword.

- Stated more succinctly, we have

$$\mathbf{G} = [\mathbf{I}_k \;\; \mathbf{P}] \quad \Longrightarrow \quad \mathbf{H} = \begin{bmatrix} \mathbf{P} \\ \mathbf{I}_{n-k} \end{bmatrix},$$

where $\mathbf{I}_m$ is the $m \times m$ identity matrix, and $\mathbf{P}$ is the remaining $k \times (n-k)$ submatrix of $\mathbf{G}$.

- **Examples.** The check matrices corresponding to (8) are

$$\mathbf{H}_{\text{parity}} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \qquad \mathbf{H}_{\text{Hamming}} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

- To give a flavor of why the check matrix is useful for decoding, notice that if $\mathbf{y} = \mathbf{x} \oplus \mathbf{z}$ with $\mathbf{z}$ indicating which bits got flipped, then

$$\begin{aligned} \mathbf{yH} &= (\mathbf{x} \oplus \mathbf{z})\mathbf{H} \\ &= \big(\mathbf{xH}\big) \oplus \big(\mathbf{zH}\big) \\ &= \mathbf{zH}, \end{aligned} \tag{9}$$

where we first used linearity, and then the fact that $\mathbf{xH} = \mathbf{0}$ for any valid codeword $\mathbf{x}$.

  - In particular, if $\mathbf{z}$ only contains a single 1 (i.e., only one bit got flipped), then $\mathbf{yH}$ is simply the i-th row of $\mathbf{H}$, where i is the index of the flipped bit.

- But notice that in $\mathbf{H}_{\text{Hamming}}$, all the rows are distinct! This means that by looking at $\mathbf{H}_{\text{Hamming}}$, we can immediately identify which bit got flipped and therefore correct it.

Notes on storage and computation.

- Notice that the code is fully specified by $\mathbf{G}$ (or $\mathbf{H}$), which only requires nk bits of storage – much smaller than the exponential storage requirement for the random coding technique used to prove the channel coding theorem!
- However, efficient decoding (getting from the noisy channel output $\mathbf{y}$ back to $\mathbf{u}$) is still challenging, designing a good choice of $\mathbf{G}$ is also difficult (but do-able!).
- In fact, at this stage it is unclear whether any good choices of G exist! This is addressed in Chapter 14 of MacKay's book (optional reading), where it is shown that a *random generator matrix* G (i.e., each entry is 1 or 0 with equal probability) achieves arbitrarily small error probability on the BSC at all rates below capacity, when used in conjunction with joint typicality decoding. The proof is very similar to the standard (non-linear) random coding proof. In summary, **very good linear codes exist**, at least if we ignore computational constraints on the decoder.

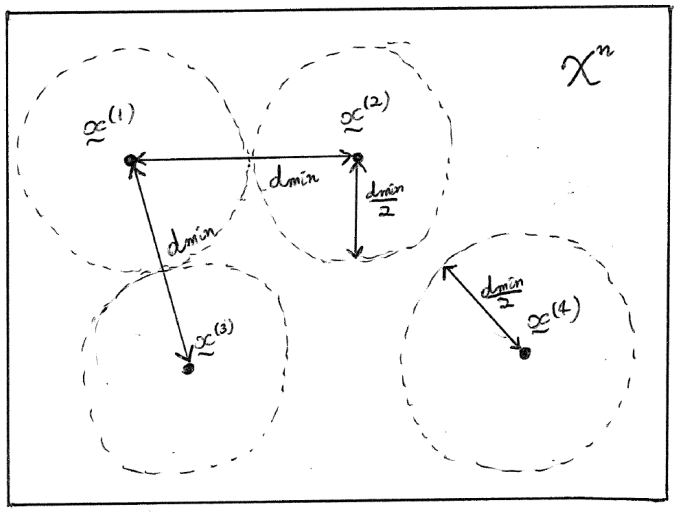
28 Distance Properties

We will only touch on the basics of distance properties here; if you are interested in knowing more, see Chapter 13 of MacKay's book.

Definition and properties.

- **Definition 1.** The *Hamming distance* between two vectors $\mathbf{x} = (x_1, \ldots, x_n)$ an $\mathbf{x}' = (x'_1, \ldots, x'_n)$ (having the same length n) is the number of positions in which they differ:
$$d_{\text{H}}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n} \mathbb{1}\{x_i \neq x'_i\}.$$
For instance, the Hamming distance between 00110011 and 00010111 is 2.
- **Definition 2.** The *minimum distance* of a codebook $\mathcal{C}$ of length-n codewords is
$$d_{\min} = \min_{\mathbf{x}\in\mathcal{C}, \mathbf{x}'\in\mathcal{C} \,:\, \mathbf{x}\neq\mathbf{x}'} d_{\text{H}}(\mathbf{x}, \mathbf{x}').$$
Intuitively, we should expect higher $d_{\min}$ to mean better robustness to noise in the channel.

- The following illustration shows that if the minimum distance is $d_{\min}$, then (at least given enough computation time) it is possible to *correct up to* $d_{\min} - 1$ *erasures* and *correct up to* $\frac{d_{\min}-1}{2}$ *bit flips*:



  - For correcting erasures: Simply note that if $d_{\min} - 1$ code symbols are replaced by an erasure symbol '?', then there is only one way to fill them in to get a valid codeword (otherwise, the minimum distance would be $d_{\min} - 1$ or less!)
  - For correcting flips: Note that the balls of radius $\frac{d_{\min}-1}{2}$ around each codeword cannot overlap. This means that if we decode each $\mathbf{y}$ to the codeword in its closest ball, we will always be correct if at most $\frac{d_{\min}-1}{2}$ flips occurred.

- **Claim.** If $\mathcal{C}$ is the set of codewords formed by a given linear code with $d_{\min} > 0$,[12] then

$$d_{\min} = \min_{\mathbf{x} \in \mathcal{C} \,:\, \mathbf{x} \neq \mathbf{0}} w(\mathbf{x}),$$

  where $w(\mathbf{x}) = \sum_{i=1}^{n} \mathbb{1}\{x_i = 1\}$ is the *weight* of the codeword $\mathbf{x}$. Hence, for linear codes, the minimum distance equals the minimum weight.

  - Proof: Let $\mathbf{x}', \mathbf{x}''$ be the two codewords at a minimum distance from each other. Then by linearity, $\mathbf{x}' \oplus \mathbf{x}''$ is also a valid codeword, and its weight is precisely $d_{\mathrm{H}}(\mathbf{x}', \mathbf{x}'') = d_{\min}$. In addition, the assumption $d_{\min} > 0$ implies that it is not the all-zero codeword.

    Conversely, since $\mathbf{x} = \mathbf{0}$ is always a valid codeword (corresponding to $\mathbf{u} = \mathbf{0}$), any codeword weight also corresponds to a distance (to the all-zero codeword).

- **Example.** Recall the 16 codewords we listed earlier for the Hamming code. The minimum non-zero weight (and hence minimum distance) is 3, which is consistent with the fact that the Hamming code can correct $\frac{3-1}{2} = 1$ bit flip.

[12] A code with $d_{\min} = 0$ is a terrible idea – it means two different vectors of information bits $\mathbf{u}, \mathbf{u}'$ lead to the same codeword!

(Optional) Distance vs. capacity.

- While a high minimum distance seems like a nice property to have, it corresponds to a fundamentally different modeling assumption compared to the channel capacity:
  - The minimum distance goal is aligned with *worst-case errors*, where (for instance) we need to be able to correct *arbitrary patterns* of up to δn bit flips introduced by the channel.
  - The channel capacity goal is aligned with *random errors*, where (for instance) we need to be able to correct bit flips that occur *independently* with probability δ each.

  Neither of these goals should be viewed as "better" than the other in general; either may be preferable depending on the application.
- Of course, it is worth asking whether achieving capacity and attaining good distance properties are actually equivalent goals. However, as argued in Chapter 13 of MacKay's book, this is not the case:
  - The best possible rate with minimum distance δn is at most the channel capacity with *double* the noise level, 2δ;
  - The channel capacity is positive for any $\delta \in \left(0, \frac{1}{2}\right)$, but no positive rate can be achieved for $\delta > \frac{1}{4}$ if one insists on a minimum distance δn;
  - Examples are known where the channel capacity is achieved despite a "bad" minimum distance.
- Although distinct from achieving capacity, the design of codes with good distance properties is a very important problem in its own right. Examples of such codes include BCH codes (a generalization of Hamming codes), and Reed-Solomon codes (commonly used for CDs and DVDs). These are based on more advanced mathematical algebraic methods, namely, polynomials on finite fields.

29 Minimum Distance Decoding

Maximum-likelihood decoding over general channels.

- Recall that in the general channel coding setup, the message m is uniform on $\{1, \dots, M\}$, the encoder maps each m to a codeword $\mathbf{x}^{(m)}$, the channel produces $\mathbf{y}$, and the error probability is given by
$$P_{\mathrm{e}} = \mathbb{P}[\hat{m} \neq m]. \tag{10}$$
where the probability is with respect to both the randomly-chosen message and the channel.

- Note: Below, we will sometimes use the message notation $m \in \{1, \ldots, M\}$ for linear codes even though it is perhaps more natural to think of the information bits $\mathbf{u} \in \{0,1\}^k$. A one-to-one correspondence between the two can be formed by considering $M = 2^k$.

- **Theorem.** For any channel $P_{\mathbf{Y}|\mathbf{X}}$ and any codebook $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)}\}$, the decoding rule that minimizes the error probability P_{e} is the maximum-likelihood (ML) decoder:

$$\hat{m} = \arg\max_{j=1,\ldots,M} P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}^{(j)}).$$

  - Proof outline: It is a standard result in Bayesian probability that the optimal estimate is the *maximum a posteriori* (MAP) estimate $\arg\max_{j=1,\ldots,M} P_{\mathbf{X}|\mathbf{Y}}(\mathbf{x}^{(j)}|\mathbf{y})$. But by Bayes' rule, we have $P_{\mathbf{X}|\mathbf{Y}}(\mathbf{x}^{(j)}|\mathbf{y}) = \frac{P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}^{(j)}) P_{\mathbf{X}}(\mathbf{x}^{(j)})}{P_{\mathbf{Y}}(\mathbf{y})}$. Since neither $P_{\mathbf{X}}(\mathbf{x}^{(j)})$ nor $P_{\mathbf{Y}}(\mathbf{y})$ depend on j (the former being because we assumed a uniformly chosen message, so all of the prior probabilities are $\frac{1}{M}$), the MAP rule is equivalent to the ML rule maximizing $P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}^{(j)})$.
  - (Note: Strictly speaking, the above outline implicitly assumes that all the codewords are distinct, but the result holds more generally, and such an assumption is very mild anyway)

Application to the BSC – Minimum distance decoding.

- For a memoryless channel, we have $P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^n P_{Y|X}(y_i|x_i)$, or equivalently $\log P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^n \log P_{Y|X}(y_i|x_i)$. For the binary symmetric channel (BSC), $P_{Y|X}(y|x_i)$ is equal to $1-\delta$ if $y = x_i$, and δ otherwise. Hence,

$$\begin{aligned}
\log P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) &= \sum_{i=1}^n \log P_{Y|X}(y_i|x_i) \\
&= \sum_{i\,:\,y_i = x_i} \log(1-\delta) + \sum_{i\,:\,y_i \neq x_i} \log \delta \\
&= (n - d_{\mathrm{H}}(\mathbf{x},\mathbf{y})) \log(1-\delta) + d_{\mathrm{H}}(\mathbf{x},\mathbf{y}) \log \delta \\
&= n \log(1-\delta) - d_{\mathrm{H}}(\mathbf{x},\mathbf{y}) \log \frac{1-\delta}{\delta}.
\end{aligned}$$

  Assuming that $\delta < \frac{1}{2}$, we have $\log \frac{1-\delta}{\delta} > 0$, and we conclude that

$$\arg\max_{j=1,\ldots,M} P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}^{(j)}) = \arg\min_{j=1,\ldots,M} d_{\mathrm{H}}(\mathbf{x}^{(j)}, \mathbf{y}).$$

  That is, maximum-likelihood decoding is equivalent to *minimum (Hamming) distance* decoding.

Application to *linear codes* for the BSC – Syndrome decoding.

- We repeat the evaluation of $\mathbf{yH}$ given above:

$$\begin{aligned}\mathbf{yH} &= (\mathbf{x} \oplus \mathbf{z})\mathbf{H} \\ &= \big(\mathbf{xH}\big) \oplus \big(\mathbf{zH}\big) \\ &= \mathbf{zH}, \end{aligned} \tag{11}$$

  with the middle step using the fact that $\mathbf{xH} = \mathbf{0}$ for any valid codeword $\mathbf{x}$.

- The quantity

$$\mathbf{S} = \mathbf{zH} = \mathbf{yH}$$

  is called the *syndrome*, and we have just shown that it can immediately be computed from the check matrix $\mathbf{H}$ given the channel output $\mathbf{y}$.

- **Claim.** For a linear code, if the syndrome is $\mathbf{S}$, then the minimum-distance codeword to $\mathbf{y}$ can be obtained by finding

$$\hat{\mathbf{z}} = \underset{\tilde{\mathbf{z}}\,:\,\tilde{\mathbf{z}}\mathbf{H}=\mathbf{S}}{\arg\min}\, w(\tilde{\mathbf{z}})$$

  (where $w(\tilde{\mathbf{z}})$ is the number of 1's in $\tilde{\mathbf{z}}$) and then computing

$$\hat{\mathbf{x}} = \mathbf{y} \oplus \hat{\mathbf{z}}.$$

  Provided that the code is systematic, the estimate $\hat{\mathbf{u}}$ of the original information bits is then formed by taking the first k entries of $\hat{\mathbf{x}}$.

  - Proof: Minimum-distance decoding searches for the codeword $\mathbf{x}^{(j)}$ $(j = 1, \ldots, M)$ having the fewest disagreements with $\mathbf{y}$. For each codeword $\mathbf{x}^{(j)}$, we can define

$$\mathbf{z}^{(j)} = \mathbf{x}^{(j)} \oplus \mathbf{y}$$

    and this clearly yields $w(\mathbf{z}^{(j)}) = d_{\mathrm{H}}(\mathbf{x}^{(j)}, \mathbf{y})$. Hence, the minimum-distance codeword is the one indexed by

$$\hat{m} = \underset{j=1,\ldots,M}{\arg\min}\, w(\mathbf{z}^{(j)}).$$

    The claim now follows since

$$\begin{aligned}\big\{\tilde{\mathbf{z}} \,:\, \tilde{\mathbf{z}}\mathbf{H} = \mathbf{S}\big\} &\overset{(a)}{=} \big\{\tilde{\mathbf{z}} \,:\, \tilde{\mathbf{z}}\mathbf{H} = \mathbf{yH}\big\} \\ &\overset{(b)}{=} \big\{\tilde{\mathbf{z}} \,:\, (\tilde{\mathbf{z}} \oplus \mathbf{y})\mathbf{H} = \mathbf{0}\big\} \\ &\overset{(c)}{=} \big\{\tilde{\mathbf{z}} \,:\, \tilde{\mathbf{z}} \oplus \mathbf{y} \text{ is a valid codeword}\big\} \\ &\overset{(d)}{=} \big\{\mathbf{x}^{(j)} \oplus \mathbf{y} \,:\, j = 1, \ldots, M\big\} \\ &\overset{(e)}{=} \big\{\mathbf{z}^{(j)} \,:\, j = 1, \ldots, M\big\}\end{aligned}$$

where (a) uses $\mathbf{S} = \mathbf{yH}$, (b) uses the definition of $\oplus$ and linearity, (c) uses the fact that $\mathbf{x}$ is a valid codeword if and only if $\mathbf{xH} = \mathbf{0}$, (d) uses the fact that the valid codewords are $\{\mathbf{x}^{(j)}\}_{j=1}^{M}$ and the fact that $\mathbf{x}^{(j)} = \tilde{\mathbf{z}} \oplus \mathbf{y}$ is equivalent to $\tilde{\mathbf{z}} = \mathbf{x}^{(j)} \oplus \mathbf{y}$, and (e) uses $\mathbf{z}^{(j)} = \mathbf{x}^{(j)} \oplus \mathbf{y}$.

- The equivalent optimization formulation in the claim can help us reduce the computational complexity of minimum-distance decoding:
  - If $\mathbf{S} = \mathbf{0}$, then simply output $\hat{\mathbf{x}} = \mathbf{y}$;
  - Otherwise, look for a $\tilde{\mathbf{z}}$ of weight 1 such that $\tilde{\mathbf{z}}\mathbf{H} = \mathbf{S}$, and if one is found, output $\hat{\mathbf{x}} = \mathbf{y} \oplus \hat{\mathbf{z}}$;
  - Otherwise, look for a $\tilde{\mathbf{z}}$ of weight 2 such that $\tilde{\mathbf{z}}\mathbf{H} = \mathbf{S}$, and if one is found, output $\hat{\mathbf{x}} = \mathbf{y} \oplus \hat{\mathbf{z}}$;
  - etc.

  If very few bit flips occurred during transmission, then we may avoid searching over an exponentially large number of codewords – there are only n vectors $\tilde{\mathbf{z}} \in \{0,1\}^n$ of weight 1, $\binom{n}{2}$ of weight 2, and so on. However, if a significant number of bit flips occur during transmission, this decoding method still typically requires far too much computation to be practical.

- **Example.** For the Hamming code (see the check matrix $\mathbf{H}_{\text{Hamming}}$ above), we could traverse the following list (starting from the top) until we find a $\tilde{\mathbf{z}}$ such that $\tilde{\mathbf{z}}\mathbf{H} = \mathbf{S}$:
  - $\tilde{\mathbf{z}} = 0000000 \implies \tilde{\mathbf{z}}\mathbf{H} = 000$
  - $\tilde{\mathbf{z}} = 1000000 \implies \tilde{\mathbf{z}}\mathbf{H} = 110$
  - $\tilde{\mathbf{z}} = 0100000 \implies \tilde{\mathbf{z}}\mathbf{H} = 101$
  - $\tilde{\mathbf{z}} = 0010000 \implies \tilde{\mathbf{z}}\mathbf{H} = 011$
  - $\tilde{\mathbf{z}} = 0001000 \implies \tilde{\mathbf{z}}\mathbf{H} = 111$
  - $\tilde{\mathbf{z}} = 0000100 \implies \tilde{\mathbf{z}}\mathbf{H} = 100$
  - $\tilde{\mathbf{z}} = 0000010 \implies \tilde{\mathbf{z}}\mathbf{H} = 010$
  - $\tilde{\mathbf{z}} = 0000001 \implies \tilde{\mathbf{z}}\mathbf{H} = 001$
  - $\tilde{\mathbf{z}} = 1100000 \implies \ldots$ (actually, in this example this case would never be reached, since the previous cases already cover all 8 possible syndromes)

  If only one bit flip occurred, the resulting $\mathbf{y} \oplus \tilde{\mathbf{z}}$ will correctly recover the transmitted codeword.

- We will not cover more advanced decoding techniques in this course; some relevant concepts are very briefly introduced in the next (optional) section, and a comprehensive reference is the book "Modern Coding Theory". MacKay's book is also a very good reference, whereas Cover/Thomas focuses almost entirely on fundamental limits rather than practical codes.

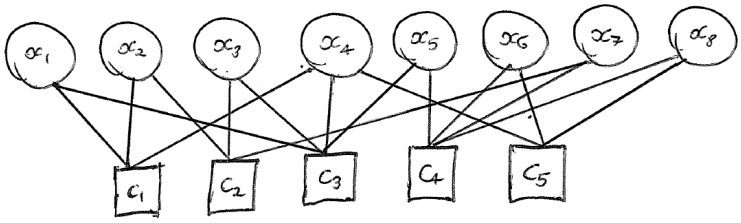
30 (Optional) Advanced Coding Methods

History.

- The channel coding theorem was published in 1948. This spawned decades of research on practical coding techniques, but none were seen to come close to achieving capacity for a long time.
- In the early 1990s, *turbo codes* were (empirically) seen to be very close to achieving capacity.
- *Low density parity check* (LDPC) codes were invented in the 1960s, but perhaps due to limited computing power, they remained unused for a long time. After the invention of turbo codes, they were rediscovered and also shown to be nearly capacity-achieving.
- *Polar codes* are a more recent development (2008). Although they usually don't perform as well as turbo/LDPC codes in practice, they are much nicer to analyze theoretically, in particular to rigorously establish that they are capacity-achieving.

(Very) Brief overview of LDPC codes.

- As a rough definition, an LDPC code is a (typically non-systematic) linear code such that the check matrix $\mathbf{H}$ contains mostly 0's and relatively few 1's.
- We can picture this via a *factor graph* representation:



  Here circles correspond to the n codeword bits ("variable nodes"), and squares correspond to the $n-k$ parity equations ("check nodes"), i.e., if there are no errors, there should be an even number of 1's connected to each square. The *low density* assumption amounts to saying that the above graph is *sparse* (has few edges).

- The reason sparsity / low density is useful is that it permits effective *belief propagation* decoding, in which information is shared between variable nodes and check nodes until accurate beliefs (posterior probability estimates) on the transmitted bits are obtained.
- Strictly speaking, this is only guaranteed to end up with accurate posterior estimates when the underlying graph has no cycles (i.e., one can never follow any path of edges and end up back at the starting node). Although sparse graphs like the ones above do have cycles, they have *very few short cycles*, and for practical purposes this is often nearly as good as having no cycles.

- For a much better introduction to LDPC codes, see Chapter 47 of MacKay's book (see also `https://www.youtube.com/watch?v=RWUxtGh-guY` for a brief introduction in Layman's terms).

(Very) Brief overview of polar codes.

- The rough idea of polar codes is as follows:
  - First consider 2 uses of the channel, $(X_1, X_2) \to (Y_1, Y_2)$.
  - The idea: Apply some pre-processing to (X_1, X_2) to create a related channel $(\widetilde{X}_1, \widetilde{X}_2) \to (Y_1, Y_2)$ such that (i) $I(X_1, X_2; Y_1, Y_2) = I(\widetilde{X}_1, \widetilde{X}_2; Y_1, Y_2)$ (no information is lost); (ii) Comparing the new channel to the original channel, one "channel use" is less noisy, and the other is more noisy.
  - Repeat this procedure several times, moving from 2 to 4 uses of the channel, then 8 uses of the channel, then 16, etc., with many "channel uses" getting less and less noisy, and the rest getting more and more noisy.
  - *For a symmetric binary channel with capacity $C \in (0, 1)$, repeating this procedure eventually leads to a fraction C of "near-perfect" channel uses, and a fraction $1 - C$ of "near-useless" channel uses.*
  - Perfect and useless channels are very easy to handle! If it were truly perfect, we could just send a bit and receive it without noise. If it were truly useless, we would simply avoid using it at all. Polar codes do something similar, sending information only over the "near-perfect" uses.
- Extensions to non-binary and non-symmetric channels were established later.
- For a much better introduction to polar codes, see the following lecture by Emre Telatar: `https://www.youtube.com/watch?v=VhyoZSB9g0w`

Lecture 8: Summary of the Course

31 Information Measures

Definitions.

- Entropy:

$$H(X) = \mathbb{E}\left[\log_2 \frac{1}{P_X(X)}\right] = \sum_x P_X(x) \log_2 \frac{1}{P_X(x)}.$$

- Joint entropy:

$$H(X,Y) = \mathbb{E}\left[\log_2 \frac{1}{P_{XY}(X,Y)}\right] = \sum_{x,y} P_{XY}(x,y) \log_2 \frac{1}{P_{XY}(x,y)}.$$

- Conditional entropy:

$$H(Y|X) = \mathbb{E}\left[\log_2 \frac{1}{P_{Y|X}(Y|X)}\right] = \sum_{x,y} P_{XY}(x,y) \log_2 \frac{1}{P_{Y|X}(y|x)} = \sum_x P_X(x) H(Y|X=x),$$

- Binary entropy function:

$$H_2(p) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p}.$$

- KL divergence (relative entropy):

$$\begin{aligned} D(P\|Q) &= \sum_x P(x) \log_2 \frac{P(x)}{Q(x)} \\ &= \mathbb{E}_{X\sim P}\left[\log_2 \frac{P(X)}{Q(X)}\right]. \end{aligned}$$

- Mutual information:

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= H(X) - H(X|Y) \\ &= D(P_{XY}\|P_X \times P_Y) \\ &= \mathbb{E}\left[\log_2 \frac{P_{XY}(X,Y)}{P_X(X)P_Y(Y)}\right] = \sum_{x,y} P_{XY}(x,y) \log_2 \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)} \\ &= \mathbb{E}\left[\log_2 \frac{P_{Y|X}(Y|X)}{P_Y(Y)}\right] = \sum_{x,y} P_{XY}(x,y) \log_2 \frac{P_{Y|X}(y|x)}{P_Y(y)}. \end{aligned}$$

- Joint mutual information:

$$I(X_1, X_2; Y_1, Y_2) = H(Y_1, Y_2) - H(Y_1, Y_2|X_1, X_2).$$

- Conditional mutual information:

$$I(X;Y|Z) = H(Y|Z) - H(Y|X,Z).$$

Properties of entropy.

- Non-negativity: $H(X) \geq 0$ with equality if and only if X is deterministic.
- Upper bound: $H(X) \leq \log_2 |\mathcal{X}|$ with equality if and only if X is uniform on $\mathcal{X}$.
- Chain rule (two variables): $H(X,Y) = H(X) + H(Y|X)$
- Chain rule (general): $H(X_1, \ldots, X_n) = \sum_{i=1}^n H(X_i|X_1, \ldots, X_{i-1})$.
- Conditioning reduces entropy: $H(X|Y) \leq H(X)$ with equality if and only if X and Y are independent.
- Sub-additivity: $H(X_1, \ldots, X_n) \leq \sum_{i=1}^n H(X_i)$ with equality if and only if $X_1, \ldots, X_n$ are independent.

Properties of mutual information.

- Non-negativity: $I(X;Y) \geq 0$ with equality if and only if X and Y are independent.
  - More generally, $D(P\|Q) \geq 0$ with equality if and only if $P = Q$
- Symmetry: $I(X;Y) = I(Y;X)$
- Upper bounds: $I(X;Y) \leq H(X) \leq \log_2 |\mathcal{X}|$ and $I(X;Y) \leq H(Y) \leq \log_2 |\mathcal{Y}|$.
- Chain rule: $I(X_1, \ldots, X_n; Y) = \sum_{i=1}^{n} I(X_i; Y | X_1, \ldots, X_{i-1})$.
- Data processing inequality: If $X \to Y \to Z$ forms a Markov chain, then $I(X;Z) \leq I(X;Y)$.

32 Symbol Source Coding

- Kraft's inequality: $\sum_{x \in \mathcal{X}} 2^{-\ell(x)} \leq 1$ (necessary and sufficient for the existence of a prefix-free code with lengths $\ell(x)$)
- Entropy bound: Average length $L(C) \geq H(X)$ for any prefix-free code
- Shannon-Fano code: Set $\ell(x) = \left\lceil \log_2 \frac{1}{P_X(x)} \right\rceil$
- Shannon-Fano code guarantee: $H(X) \leq L(X) < H(X) + 1$.
- Huffman code is the optimal symbol code
- Limitations: (i) +1 bit can be significant; (ii) Symbol codes do not exploit memory in the source, so their fundamental limit $\sum_{i=1}^{n} H(X_i)$ may be much higher than $H(X_1, \ldots, X_n)$.

33 Block Source Coding

- Notation: Discrete memoryless source $P_{\mathbf{X}}(\mathbf{x}) = \prod_{i=1}^{n} P_X(x_i)$, block length n, rate R, number of encoded messages $M = 2^{nR}$, error probability $P_{\mathrm{e}} = \mathbb{P}[\hat{\mathbf{X}} \neq \mathbf{X}]$.
- Source coding theorem: There exist codes with P_{e} being arbitrarily small when $R > H(X)$, but no such codes exist when $R < H(X)$.
- Achievability via typical sets:
  - Typical set $\mathcal{T}_n(\epsilon) = \left\{\mathbf{x} \in \mathcal{X}^n \,:\, 2^{-n(H(X)+\epsilon)} \leq P_{\mathbf{X}}(\mathbf{x}) \leq 2^{-n(H(X)-\epsilon)}\right\}$.
  - $\mathbb{P}[\mathbf{X} \in \mathcal{T}_n(\epsilon)] \to 1$ as $n \to \infty$
  - $|\mathcal{T}_n(\epsilon)| \leq 2^{n(H(X)+\epsilon)}$ and $|\mathcal{T}_n(\epsilon)| \geq (1 - o(1))2^{n(H(X)-\epsilon)}$
  - Asymptotic equipartition property: The source is asymptotically uniformly distributed (on the typical set).

- Can perform compression by assigning each typical sequence a unique message index (possible if $R > H(X)$).

- Converse via Fano's inequality:
  - Fano's inequality: $H(X|\hat{X}) \leq H_2(P_\mathrm{e}) + P_\mathrm{e} \log_2\left(|\mathcal{X}| - 1\right)$ where $P_\mathrm{e} = \mathbb{P}[\hat{X} \neq X]$
  - Applied to sequences: $H(\mathbf{X}|\hat{\mathbf{X}}) \leq H_2(P_\mathrm{e}) + P_\mathrm{e} \log_2\left(|\mathcal{X}|^n - 1\right)$ where $P_\mathrm{e} = \mathbb{P}[\hat{\mathbf{X}} \neq \mathbf{X}]$
  - Weakened and re-arranged form: $P_\mathrm{e} \geq \frac{1}{\log_2|\mathcal{X}|}\left(H(X) - \frac{1}{n} I(\mathbf{X}; \hat{\mathbf{X}}) - \frac{1}{n}\right)$
  - Use $I(\mathbf{X}; \hat{\mathbf{X}}) \leq H(\hat{\mathbf{X}}) \leq nR$ to complete the converse proof

34 Channel Coding

- Notation: Discrete memoryless channel $P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^n P_{Y|X}(y_i|x_i)$, block length n, rate R, number of messages $M = 2^{nR}$, error probability $P_\mathrm{e} = \mathbb{P}[\hat{m} \neq m]$.

- Channel coding theorem: There exist codes with P_e being arbitrarily small when $R < C$, but no such codes exist when $R > C$, where

$$C = \max_{P_X} I(X; Y).$$

- Examples:
  - Noiseless binary channel: $C = 1$
  - Binary symmetric channel: $C = 1 - H_2(\delta)$
  - Binary erasure channel: $C = 1 - \epsilon$

- Achievability via random coding and joint typicality:
  - **Typical set**: Pairs $(\mathbf{x}, \mathbf{y})$ satisfying

$$\begin{aligned} 2^{-n(H(X)+\epsilon)} &\leq P_{\mathbf{X}}(\mathbf{x}) \leq 2^{-n(H(X)-\epsilon)} \\ 2^{-n(H(Y)+\epsilon)} &\leq P_{\mathbf{Y}}(\mathbf{y}) \leq 2^{-n(H(Y)-\epsilon)} \\ 2^{-n(H(X,Y)+\epsilon)} &\leq P_{\mathbf{XY}}(\mathbf{x}, \mathbf{y}) \leq 2^{-n(H(X,Y)-\epsilon)}. \end{aligned}$$

  - High probability property: $\mathbb{P}[(\mathbf{X}, \mathbf{Y}) \in \mathcal{T}_n(\epsilon)] \to 1$ as $n \to \infty$.
  - Size of jointly typical set: $|\mathcal{T}_n(\epsilon)| \leq 2^{n(H(X,Y)+\epsilon)}$.
  - Probability of independent sequences being jointly typical: $\mathbb{P}[(\mathbf{X}', \mathbf{Y}') \in \mathcal{T}_n(\epsilon)] \leq 2^{-n(I(X;Y)-3\epsilon)}$.
  - **Random coding:** Generate every symbol of every codeword independently according to some distribution P_X

- Joint typicality decoding: Look for a unique codeword jointly typical with the received $\mathbf{y}$
- Error probability upper bound:

$$\overline{P}_{\mathrm{e}}^{(m)} \leq \mathbb{P}\big[(\mathbf{X}^{(m)}, \mathbf{Y}) \notin \mathcal{T}_n(\epsilon)\big] + \sum_{\tilde{m} \neq m} \mathbb{P}\big[(\mathbf{X}^{(\tilde{m})}, \mathbf{Y}) \in \mathcal{T}_n(\epsilon)\big].$$

- Apply $\mathbb{P}[(\mathbf{X}, \mathbf{Y}) \in \mathcal{T}_n(\epsilon)] \to 1$ to first term, $\mathbb{P}[(\mathbf{X}', \mathbf{Y}') \in \mathcal{T}_n(\epsilon)] \leq 2^{-n(I(X;Y)-3\epsilon)}$ to second term.

- Converse via Fano's inequality:
  - Fano's inequality: $H(m|\hat{m}) \leq H_2(P_{\mathrm{e}}) + P_{\mathrm{e}} \log_2(M-1)$
  - Re-arranged form: $P_{\mathrm{e}} \geq 1 - \frac{I(m;\hat{m})+1}{\log_2 M}$.
  - Apply data processing inequality to get $I(m;\hat{m}) \leq I(\mathbf{X};\mathbf{Y})$, and then establish that $I(\mathbf{X};\mathbf{Y}) \leq \sum_{i=1}^{n} I(X_i;Y_i)$ (due to discrete memoryless channel assumptions)

35 Practical Channel Codes

Definitions.

- Parity check of m bits: $c = b_1 \oplus \ldots \oplus b_m$
- Linear code: Every codeword bit in $\mathbf{x} = (x_1, \ldots, x_n)$ is a parity check of some subset of the message bits $\mathbf{u} = (u_1, \ldots, u_k)$
  - Linearity property: If $\mathbf{x}'$ and $\mathbf{x}''$ are codewords, then so is $\mathbf{x}' \oplus \mathbf{x}''$
- Number of messages 2^k, rate $\frac{k}{n}$
- Generator matrix: $\mathbf{x} = \mathbf{uG}$ for some $\mathbf{G} \in \{0,1\}^{k \times n}$ whose columns put 1's for the bits included in that parity check
- Systematic code: $\mathbf{G} = [\mathbf{I}_k \;\; \mathbf{P}]$ (i.e., first k codeword bits are direct copies of the message bits)
- Check matrix for a systematic code: $\mathbf{H} = \begin{bmatrix} \mathbf{P} \\ \mathbf{I}_{n-k} \end{bmatrix}$
  - Key property: $\mathbf{xH} = \mathbf{0} \iff \mathbf{x}$ is a codeword

Hamming distance and decoding.

- Hamming distance: $d_{\mathrm{H}}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n} \mathbb{1}\{x_i \neq x_i'\}$.
- Minimum distance of a code: $d_{\min} = \min_{\mathbf{x} \in \mathcal{C}, \mathbf{x}' \in \mathcal{C} : \mathbf{x} \neq \mathbf{x}'} d_{\mathrm{H}}(\mathbf{x}, \mathbf{x}')$.
- At least in the absence of computational limitations, one can always correct up to $d_{\min} - 1$ erasures, or up to $\frac{d_{\min}-1}{2}$ errors
- For a linear code, $d_{\min} = \min_{\mathbf{x} \in \mathcal{C} : \mathbf{x} \neq \mathbf{0}} w(\mathbf{x})$, where $w(\mathbf{x}) = \sum_{i=1}^{n} \mathbb{1}\{x_i = 1\}$ is the number of 1's in $\mathbf{x}$
- Maximum likelihood decoding (optimal for general codes and channels):

$$\hat{m} = \arg\max_{j=1,\ldots,M} P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}^{(j)}).$$

- Minimum distance decoding (optimal for general codes for the BSC):

$$\hat{m} = \arg\min_{j=1,\ldots,M} d_{\mathrm{H}}(\mathbf{x}^{(j)}, \mathbf{y}).$$

- Syndrome decoding (a sometimes-efficient implementation of minimum distance decoding): (i) Compute the syndrome $\mathbf{S} = \mathbf{zH} = \mathbf{yH}$, (ii) Find $\hat{\mathbf{z}} = \arg\min_{\tilde{\mathbf{z}} : \tilde{\mathbf{z}}\mathbf{H} = \mathbf{S}} w(\tilde{\mathbf{z}})$; (iii) Set $\hat{\mathbf{x}} = \mathbf{y} \oplus \hat{\mathbf{z}}$; (iv) Set $\hat{m}$ to be the message that produces $\hat{\mathbf{x}}$ (trivial if the code is systematic).

- Can work from lowest weights upwards:
  - If $\mathbf{S} = \mathbf{0}$, then $\mathbf{y}$ is already a valid codeword, so let $\hat{\mathbf{x}} = \mathbf{y}$ (corresponding to $\tilde{\mathbf{z}} = \mathbf{0}$)
  - Otherwise, look for a $\tilde{\mathbf{z}}$ of weight 1 such that $\tilde{\mathbf{z}}\mathbf{H} = \mathbf{S}$, and if one is found, output $\hat{\mathbf{x}} = \mathbf{y} \oplus \hat{\mathbf{z}}$;
  - Then continue to weight 2, weight 3, etc. as needed.

36 Interesting Topics that we Didn't Cover

- Sources (and channels) with memory
  - ...and practical coding methods for them (Arithmetic coding, Lempel-Ziv coding)
- Most practical channel coding methods and efficient decoders
  - Turbo codes, LDPC codes, polar codes, BCH codes, Reed-Solomon codes, Reed-Muller codes, etc.
- Multi-user communication
  - Multiple-access channel, broadcast channel, Slepian-Wolf coding, etc.
- 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

Lecture 9: Notes on Reed–Solomon Codes

[][Rahul Jain][Mark Haiman] Reed-Solomon codes are examples of error correcting codes, in which redundant information is added to data so that it can be recovered reliably despite errors in transmission or storage and retrieval. The error correction system used on CD's and DVD's is based on a Reed-Solomon code. These codes are also used on satellite links and other communications systems.

We will first discuss the general principles behind error correcting codes. Then we will explain one type of Reed-Solomon code, and discuss the algorithms for encoding using this code, and for decoding with error correction.

1. Error correcting and error detecting codes

In an error correcting code, a message M is encoded as a sequence of symbols $a_1 a_2 \dots a_n$, called a codeword. The set of possible symbols is fixed in advance; for instance each symbol might be a byte (8 bits) of binary data. The code incorporates some redundancy, so that if some of the symbols in a codeword are changed, we can still figure out what the original message must have been. Usually, the number of symbols in the codeword, n, is also fixed, so each message carries a fixed amount of data. To encode larger amounts of data, one would break it up into a number of messages and encode each one separately.

The simplest example of an error correcting code is the triple-redundancy code. In this code, the message M consists of a single symbol a, and we encode it by repeating the symbol three times, as aaa. Suppose one symbol in the codeword is changed, so we receive a word baa or aba or aab. We can still recover the original symbol a by taking a majority vote of the three symbols received. If errors result in two symbols being changed, we might receive a word like abc. In that case we can't correct the errors and recover the original message, but we can at least detect the fact that errors occurred. However, with this code we cannot always detect two errors. If the code word was aaa, two errors might change it to something like bba. This received word would be miscorrected by the majority vote decoding algorithm, giving the wrong message symbol b.

By repeating the message more times, we can achieve higher rates of error correction: with five repetitions we can correct two errors per codeword, and with seven repetitions we can correct three. These simple-minded error correcting codes are very inefficient, however,

since we have to store or transmit three, five or seven times the amount of data in the message itself. Using more sophisticated mathematical techniques, we will be able to achieve greater efficiency.

1.1. Hamming distance. An obvious strategy for decoding any error correcting code is, if we receive a word B that is not a codeword, to find the codeword A that is "closest" to B, and then decode A to get the message. For example, majority-vote decoding in the triple-redundancy code follows this strategy. If we receive a word such as baa, the closest codeword is aaa, since it differs from the received word in only one place, while any other codeword xxx differs from the

Date: February, 2003.

received word baa in at least two places. It is useful to make a precise definition of this concept of "closeness."

Definition 1. The Hamming distance $d(A, B)$ between two words $A = a_1 a_2 \dots a_n$ and $B = b_1 b_2 \dots b_n$ is the number of places where they differ. In other words $d(A, B)$ is the number of indices i such that $a_i \neq b_i$.

The Hamming distance satisfies the following inequality, called the triangle inequality (because it is also satisfied by the lengths of the sides of a triangle):

$$d(A, C) \leq d(A, B) + d(B, C) \quad \text{for all words } A, B, C \tag{1}$$

To see why the triangle inequality holds, observe that we can change A into B by altering $d(A, B)$ symbols, and then change B into C by altering another $d(B, C)$ symbols. So we can change A into C by altering at most $d(A, B) + d(B, C)$ symbols (perhaps fewer, since in changing B to C we might undo some of the changes from A to B).

The standard decoding algorithm is as follows. Given a received word B, find the codeword or codewords A that minimize the Hamming distance $d(A, B)$. If there is just one such codeword A, decode it to get the message. If there is more than one such codeword, report that an uncorrectable error has been detected. Our first theorem tells us how many errors a code can correct or detect using the standard decoding algorithm.

Theorem 1. Let d be the minimum Hamming distance between pairs of distinct codewords of a code C. If $d \geq 2e + 1$, the standard decoding algorithm for C can correct up to e errors. If $d \geq 2e + 2$, it can correct up to e errors and detect $e + 1$ errors.

Proof. Suppose we cannot correct up to e errors. This means there is a codeword A, and some combination of at most e errors that changes A to a word B which is either miscorrected or uncorrectable. The fact that we can change A to B with at most e errors means that $d(A, B) \leq e$. If B is miscorrected or uncorrectable, it means there is a word $C \neq A$ with $d(C, B) \leq d(A, B)$, and therefore $d(C, B) \leq e$. By the triangle inequality, we have $d(A, C) \leq 2e$, contradicting the hypothesis that $d \geq 2e + 1$.

For the second part of the theorem, observe that if $d \geq 2e + 2$ then we can correct up to e errors by the first part. Suppose that we cannot also detect $e + 1$ errors. Then there is a codeword A and a combination of $e + 1$ errors that changes A to a word B which will be miscorrected. For that to happen there must be a codeword C with $d(B, C) \leq e$. Since $d(A, B) = e + 1$, the triangle inequality gives $d(A, C) \leq 2e + 1$, contradicting the hypothesis

$d \geq 2e + 2$.

Example. Consider the n-fold redundancy code, where the message is a single symbol a, and the corresponding codeword is n copies $aa \dots a$. The Hamming distance between any two distinct codewords is $d(aa \dots a, bb \dots b) = n$, so the minimum distance d for this code is equal to n. If n is odd, equal to $2e + 1$, then we can correct e errors, as we have seen before: triple-redundancy corrects one error, quintuple redundancy corrects two, and so on. If n is even, equal to $2e + 2$, then we can also detect $e + 1$ errors. So the quadruple redundancy code can correct one error, just like the triple redundancy code, but it can also detect any two errors, which the triple redundancy code cannot do.

There are several practical difficulties in using Theorem 1 to construct error correcting codes. The first difficulty is that of determining the minimum Hamming distance d between codewords.

For a completely arbitrary code, the only way to do this is by comparing every two codewords. However, for practical applications, we want to design codes in which a substantial amount of data is contained in each message. This means that the number of possible messages, and therefore the number of possible codewords, will be enormous. It then becomes impractical to find the minimum Hamming distance by comparing all possible pairs of codewords.

A second difficulty, which is related to the first one, is that we need an efficient means of encoding. If our code is completely arbitrary, the only way to encode is by means of a table listing the codeword for every message. Once again, if we wish to design a code in which each message message contains a substantial amount of data, and so there are a huge number of possible messages, it becomes impractical to keep a table of codewords for all the possible messages.

The third difficulty is that the standard decoding algorithm is impractical to carry out for an arbitrary code. If we receive a word B that is not a codeword, and we want to locate the closest codeword A, the best we can do is to first try every possible one-symbol change to B, then every possible two-symbol change, and so on, until we find one or more codewords. This could require a very long search to decode just one single message.

The difficulty of encoding is easily overcome with the help of linear codes, which we will discuss in the next section. However, not all linear codes help much with the difficulties involved in determining the minimum distance and efficient decoding. For that, we need codes with even more special properties, known generally as algebraic codes. The Reed-Solomon codes, which we will study in the last sections, are examples of algebraic codes.

2. LINEAR CODES

For the linear codes we are going to study, the code symbols a will be integers $(\mathrm{mod} p)$, for a fixed prime number p. We use the notation

$$\mathbb{Z}_p = \text{ set of all congruence classes } (\mathrm{mod} p).$$

To be definite, we represent every integer by its remainder $(\bmod p)$, so there are p different symbols $a \in \mathbb{Z}_p$, represented by the p numbers $0, 1, 2, \dots, p - 1$. Using modular arithmetic,

we can add, subtract, and multiply these symbols. Because p is prime, every symbol $a \neq 0$ has a multiplicative inverse a^{-1}, so we can also "divide" symbols.

The set $\mathbb{Z}_p$ with its arithmetic operations is an example of what mathematicians call a field: a system with addition, subtraction, multiplication, and division (except by zero) obeying the usual identities of algebra. Since it is a finite set, $\mathbb{Z}_p$ is a finite field. In more advanced algebra courses, other examples of finite fields are constructed, which may also be used as the set of symbols for a linear code. It turns out that there is a finite field with p^d elements for every prime number p and every positive integer d. The Reed-Solomon codes most commonly encountered in practice are based on the field with $2^8 = 256$ elements, so the symbols can be conveniently represented by 8 -bit binary bytes. To keep things simple, in these notes we will only study codes based on $\mathbb{Z}_p$. However, the same concepts apply to codes based on other finite fields.

A $1 \times n$ matrix - that is, a matrix with one row-is called a row vector. A word $a_1 a_2 \dots a_n$ with symbols $a_i \in \mathbb{Z}_p$, can (and will) be represented by the row vector

$$\begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix}.$$

Definition 2. Fix positive integers $m < n$, and an $m \times n$ matrix $\mathbf{C}$ with entries in $\mathbb{Z}_p$. The code in which a message vector $\begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix}$ is encoded by the code vector

$$\begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix} \cdot \mathbf{C}$$

is called a linear code with m message symbols and n codeword symbols.

For a linear code to be any use at all, we must be sure that the encoding function from message vectors $\mathbf{x}$ to code vectors $\mathbf{a} = \mathbf{xC}$ is one-to-one, that is, we can recover the message from the code. One way to ensure this is to use a matrix $\mathbf{C}$ that is in normal form, meaning that the first m columns of $\mathbf{C}$ form an $m \times m$ identity matrix:

$$\mathbf{C} = [\mathbf{I}_m \mid \mathbf{P}],$$

where $\mathbf{P}$ is an $m \times (n - m)$ matrix. Then from the rules of matrix multiplication we deduce that the message $\mathbf{a} = \text{xC}$ will have the form $\mathbf{a} = [\mathbf{x} \mid \mathbf{c}]$, where $\mathbf{c} = \text{xP}$. In other words, the codeword is just the message word, follow by some additional check symbols $\mathbf{c}$. In this case, we can obviously recover $\mathbf{x}$ from a if there are no errors, just by looking at the first m symbols.

One important point about linear codes is that by their very definition, there is an easy algorithm for encoding a message: just multiply it by the code matrix C. A second important point is that for linear codes there is a theorem that allows us to determine the minimum Hamming distance between codewords without checking every possible pair. Before giving the theorem, we first need the following lemma, which expresses a key property of linear codes.

Lemma 1. If $\mathbf{a}$ and $\mathbf{b}$ are two codewords in a linear code C, then $\mathbf{a} + \mathbf{b}$ and $\mathbf{a} - \mathbf{b}$ are also codewords, and more generally so is $c\mathbf{a} + d\mathbf{b}$, for any scalars c, d in the base field $\mathbb{Z}_p$.

Proof. By definition, there exist message vectors $\mathbf{x}$ and $\mathbf{y}$ such that $\mathbf{a} = \text{xC}$ and $\mathbf{b} = \mathbf{yC}$. Then from the rules of matrix multiplication we see that

$$c\mathbf{a} + d\mathbf{b} = (c\mathbf{x} + d\mathbf{y})\mathbf{C}.$$

Now $c\mathbf{x} + d\mathbf{y}$ is again a $1 \times m$ row vector, and thus a possible message vector, so therefore $c\mathbf{a} + d\mathbf{b}$ is a code vector. To get the special cases $\mathbf{a} \pm \mathbf{b}$ take $c = 1$ and $d = \pm 1$.

Now for the theorem about Hamming distance.

Theorem 2. The minimum Hamming distance d between codewords in a linear code C is equal to the minimum number of non-zero symbols in a code vector $\mathbf{a} \neq \mathbf{0}$ of C.

Proof. Let $\mathbf{b}$ and $\mathbf{c}$ be distinct code vectors with minimum distance $d(\mathbf{b}, \mathbf{c}) = d$. Then $\mathbf{a} = \mathbf{b} - \mathbf{c}$ is a code vector, by Lemma 1 , and the number of non-zero symbols in a is equal to d. This shows that the minimum number of non-zero symbols in a code vector is less than or equal to d. To prove the opposite inequality, note that Lemma 1 implies that $\mathbf{0}$ is a code vector. For every code vector $\mathbf{a}$, we have $d(\mathbf{a}, \mathbf{0})$ equal to the number of non-zero symbols in $\mathbf{a}$, so this number is greater than or equal to d.

Definition 3. The weight of a code vector is the number of non-zero symbols in it. The weight of a linear code is the minimum weight of a non-zero code vector. Thus Theorem 2 says that the minimum Hamming distance in a linear code is equal to its weight. Combining this with Theorem 1, we see that a linear code with odd weight $2e + 1$ can correct e errors, and a linear code with even weight $2e + 2$ can correct e errors and detect $e + 1$ errors.

3. Reed-Solomon codes

All linear codes offer the advantages that they are easy to encode, and the minimum Hamming distance reduces to a simpler concept, the weight. However, the weight of an arbitrary linear code is still not easy to determine. Also, the mere fact that a code is linear doesn't help all that much in applying the standard decoding algorithm. In order to compute the weight and decode more easily, we need to use linear codes with special properties. Usually the special properties are based on algebra, in which case the code is called an algebraic code. The Reed-Solomon codes that we will now define are examples of algebraic codes.

Definition 4. Let p be a prime number and let $m \leq n \leq p$. The Reed-Solomon code over the field $\mathbb{Z}_p$ with m message symbols and n code symbols is defined as follows. Given a message vector $\begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix}$, let $P(t)$ be the polynomial

$$P(t) = x_m t^{m-1} + x_{m-1} t^{m-2} + \cdots + x_2 t + x_1$$

with coefficients given by the message symbols. Thus $P(t)$ is a polynomial of degree at most $m - 1$ in one variable t, with coefficients in $\mathbb{Z}_p$. Then the code vector a for this message vector is the list of the first n values of the polynomial $P(t)$:

$$\mathbf{a} = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix} = \begin{bmatrix} P(0) & P(1) & \dots & P(n-1) \end{bmatrix}$$

(evaluated using modular arithmetic in $\mathbb{Z}_p$).

From this definition it might not be obvious that Reed-Solomon codes are in fact linear codes.

However, using the definition of matrix multiplication, we can deduce the following linear presentation of the Reed-Solomon code.

Theorem 3. The Reed-Solomon code over $\mathbb{Z}_p$ with m message symbols and n code symbols is the linear code with matrix

$$\mathbf{C} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 0 & 1 & 2 & \cdots & n-1 \\ 0^2 & 1^2 & 2^2 & \cdots & (n-1)^2 \\ \vdots & \vdots & \vdots & & \vdots \\ 0^{m-1} & 1^{m-1} & 2^{m-1} & \dots & (n-1)^{m-1} \end{bmatrix}$$

all entries taken (mod$p$).

Although the code matrix $\mathbf{C}$ for a Reed-Solomon code is not in normal form, we will see shortly that it does define a one-to-one encoding function.

Next we want to compute the weight of a Reed-Solomon code. To do this we first have to recall some facts about polynomials that you learned in high school algebra. Of course, when you learned algebra in high school, you learned about polynomials with coefficients in the field of real numbers (or maybe the field of rational numbers). Now we are working with polynomials that have coefficients in $\mathbb{Z}_p$. However, what you learned in high school is still valid in this situation, because $\mathbb{Z}_p$ is a field, and the facts we will now recall depend only on laws of algebra that hold in every field. Here they are.

Lemma 2. If a is a root of a polynomial $P(t)$, that is, if $P(a) = 0$, then $P(t)$ can be factored as $P(t) = (t-a)Q(t)$. Note that the degree of $Q(t)$ is one less than the degree of $P(t)$.

Proof. You probably learned the proof of this in high school, although you might not have realized at the time that what you were learning was the proof of a theorem. I'll just remind you quickly how it works. Regardless of whether or not a is a root of $P(t)$, you can always divide $P(t)$ by $(t-a)$ using long division, obtaining a quotient polynomial $Q(t)$ and a remainder c, which is a constant. Then $P(t) = (t-a)Q(t) + c$, and substituting $t = a$ in this identity shows that $c = P(a)$. So if a is a root of $P(t)$ then $c = 0$, and the result follows.

Theorem 4. If $P(t)$ has d distinct roots and is not identically zero, then $P(t)$ has degree at least d.

Proof. Suppose $a_1, \dots, a_d$ are distinct roots of $P(t)$. By Lemma 2, we can factor $P(t) = (t - a_d)\,Q_1(t)$, and substituting $t = a_i$ into this for $i \neq d$ shows that $a_1, \dots, a_{d-1}$ are roots of $Q_1(t)$. In turn we can factor $Q_1(t) = (t - a_{d-1})\,Q_2(t)$, where $a_1, \dots, a_{d-2}$ are roots of $Q_2(t)$. Repeating this process we eventually get $P(t) = (t - a_d)\,(t - a_{d-1}) \cdots (t - a_1)\,Q_d(t)$. If $P(t)$ is not identically zero, then neither is $Q_d(t)$, so the factorization shows that $P(t)$ has degree at least d.

Now we are ready to determine the weights of our codes.

Theorem 5. The weight of the Reed-Solomon code over $\mathbb{Z}_p$ with m message symbols and n code symbols is equal to $n - m + 1$.

Proof. First we'll show that the weight is at least $n - m + 1$. By the definition of the Reed-Solomon code, this is equivalent to saying that for every non-zero polynomial $P(t)$ of

degree at most $m-1$, when we evaluate it at the elements $0, 1, \ldots, n-1$ of $\mathbb{Z}_p$, we will get at least $n-m+1$ non-zero values. Supposing to the contrary that we get at most $n-m$ non-zero values, we see that $P(t)$ has at least m roots among the elements $0, 1, \ldots, n-1$. Note that $n \leq p$ by definition, so these roots are distinct elements of $\mathbb{Z}_p$. But by Theorem 4 this is impossible if $P(t)$ has degree less than m and is not identically zero.

To show that the weight is at most $n-m+1$ and complete the proof, we just have to find an example of a code vector with weight exactly $n-m+1$. Again by the definition of the code, this is equivalent to finding an example of a polynomial $P(t)$ with degree at most $m-1$ such that exactly $m-1$ of the values $P(0), \ldots, P(n-1)$ are zero. An example of such a polynomial is $P(t) = t(t-1)(t-2)\cdots(t-(m-2))$.

Corollary 1. A Reed-Solomon code with m message symbols and $n = m+2e$ code symbols can correct e errors.

Example. A Reed-Solomon code with 15 code symbols can transmit 11 message symbols, while correcting errors in any two of the code symbols. By contrast, using a quintuple redundancy code to achieve two-error correction, we can only transmit 3 message symbols with 15 code symbols.

To complete up our discussion of encoding, we will now show, as promised earlier, that the encoding function is one-to-one. This amounts to saying that if two message polynomials $P_1(t)$ and $P_2(t)$ give the same code vector, that is, if

$$P_1(i) = P_2(i) \text{ for } i = 0, 1, \ldots, n-1,$$

then we must have $P_1(t) = P_2(t)$ identically. To see this, observe that $P_1(t) - P_2(t)$ is a polynomial of degree less than m, and it has at least n distinct roots in $\mathbb{Z}_p$, namely $0, 1, \ldots, n-1$. Since $m \leq n$ by definition, we must have $P_1(t) - P_2(t) = 0$, by Theorem 4 .

4. Decoding Reed-Solomon codes

What makes Reed-Solomon codes really useful is the existence of a simple algorithm for correcting errors and decoding. The algorithm we will discuss was discovered by Berlekamp and Welch. Rather than publish it in a mathematical or engineering journal, they patented it in 1983. More on that at the end.

The discussion in this section will always refer to the Reed-Solomon code over $\mathbb{Z}_p$ with

$$\begin{aligned} m &= \text{ number of message symbols ,} \\ n &= \text{ number of code symbols} \\ &= m + 2e, \\ e &= \text{ number of errors the code can correct.} \end{aligned}$$

The decoding procedure relies on two lemmas, which we can prove using Theorem 4.
Lemma 3. Let the transmitted code vector be

$$\begin{bmatrix} P(0) & P(1) & \ldots & P(n-1) \end{bmatrix}$$

and the received vector be

$$\left[\begin{array}{cccc} R_0 & R_1 & \ldots & R_{n-1} \end{array}\right]$$

Assume there are at most e errors, that is, at most e values i such that $R_i \neq P(i)$. Then there exist non-zero polynomials

$$\begin{aligned} E(t) \text{ of degree } &\leq e \\ Q(t) \text{ of degree } &\leq m + e - 1, \end{aligned}$$

such that

$$Q(i) = R_i E(i) \quad \text{for all } i = 0, 1, \ldots, n \tag{2}$$

Proof. Let $\{i_1, i_2, \ldots, i_k\}$ be the set of error positions, that is, i is a member of this set if $R_i \neq P(i)$. Let

$$\begin{aligned} E(t) &= (t - i_1)\,(t - i_2) \cdots (t - i_k)\,, \\ Q(t) &= P(t)E(t) \end{aligned}$$

Clearly $E(t)$ has degree k, which is less than or equal to e by assumption. Since $P(t)$ has degree $\leq m - 1$, it also follows that $Q(t)$ has degree $\leq m + e - 1$.

To see why equation (2) holds for every i, consider two cases. The first case is that i is not an error position. Then $R_i = P(i)$, so $Q(i) = P(i)E(i) = R_i E(i)$. The second case is that i is an error position, so it belongs to the set $\{i_1, i_2, \ldots, i_k\}$. From our definition of $E(t)$ we see that in this case $E(i) = 0$, and therefore $Q(i) = P(i)E(i) = 0 = R_i E(i)$. So equation (2) holds in either case.

Equation (2) is called the key equation for the decoding algorithm. Lemma 3 tells us that it has a non-zero solution. However, the solution we wrote down to prove the lemma can only be calculated explicitly if we already know the error positions. In practice, we have to solve the key equation by interpreting it as a system of linear equations for the unknown coefficients of the polynomials $E(t)$ and $Q(t)$. The lemma guarantees that a non-zero solution of this linear system must exist.

The next lemma shows that after solving the key equation, we can calculate the original message polynomial $P(t)$.

Lemma 4. If $E(t)$ and $Q(t)$ satisfy the key equation (2) in Lemma 3, and the number of errors is at most e, then $Q(t) = P(t)E(t)$. Hence we can compute $P(t)$ as $Q(t)/E(t)$.

Proof. In the course of proving Lemma 3 we showed that there exists a solution of (2) with $Q(t) = P(t)E(t)$, but now we must show that every solution has this property.

Observe that both $Q(t)$ and $P(t)E(t)$ are polynomials of degree $\leq m + e - 1$, and hence so is their difference $Q(t) - P(t)E(t)$. If i is not an error position, then $P(i) = R_i$, and therefore $Q(i) - P(i)E(i) = 0$. There are at least $n - e$ non-error positions, so the polynomial $Q(t) - P(t)E(t)$ has at least $n - e = m + e$ distinct roots in $\mathbb{Z}_p$. Since its degree is less than $m + e$, we must have $Q(t) - P(t)E(t) = 0$ identically, by Theorem 4 .

Based on these two lemmas we can describe the decoding algorithm.
Decoding algorithm. Suppose the received vector is

$$\begin{bmatrix} R_0 & R_1 & \dots & R_{n-1} \end{bmatrix}$$

Introduce variables $u_0, u_1, \dots, u_{m+e-1}$ and $v_0, v_1, \dots, v_e$ to stand for the coefficients of $Q(t)$ and $E(t)$, so

$$Q(t) = u_{m+e-1}t^{m+e-1} + u_{m+e-2}t^{m+e-2} + \cdots + u_1 t + u_0$$
$$E(t) = v_e t^e + v_{e-1}t^{e-1} + \cdots + v_1 t + v_0$$

For each $i = 0, 1, \dots, n-1$, substitute $t = i$ in $Q(t)$ and $E(t)$ to evaluate the key equation

$$Q(i) = R_i E(i).$$

This results in a system of n linear equations for the unknown coefficients u_j and v_k. Find a nonzero solution of this system, and plug in the result to get $Q(t)$ and $E(t)$ explicitly. Finally, divide $Q(t)$ by $E(t)$ to recover the message polynomial $P(t)$.

What might go wrong if there are more than e errors in the received vector? The first main step in the algorithm is to solve n linear equations for the $n+1$ unknowns u_j and v_k. It is a general theorem of linear algebra that this is always possible. The second main step is to divide $Q(t)$ by $E(t)$ to find $P(t)$. On doing this we might find that there is a remainder, and therefore no solution of $Q(t) = P(t)E(t)$. If that occurs, then Lemma 4 implies that there must have been more than e errors. We cannot correct them, but at least we have detected them. Not every combination of more than e errors can be detected, of course - some will simply result in incorrect decoding.

5. An improvement to the decoding algorithm

Most of the work in the decoding algorithm lies in solving n linear equations for the $n+1$ unknowns u_j and v_k, a straightforward but fairly complex task that must be repeated for every message vector. In practice one often wants to decode data at high rates, so a reduction in the complexity per message is desirable. In this section we will see how to use some ideas from polynomial interpolation theory to simplify decoding.

The general theory described in the previous sections can be applied to Reed-Solomon codes over any finite field, but the method I will now describe is a special one for the fields $\mathbb{Z}_p$. We begin with a classical technique for predicting all the values of a polynomial function from the first few.

Definition 5. The first difference of a sequence

$$a_1, a_2, a_3, \dots$$

is the sequence

$$\Delta a_i = a_{i+1} - a_i.$$

The first difference of the first differences is called the second difference and denoted $\Delta^2 a_i$; similarly we have third differences $\Delta^3 a_i$ and so on.

Here is an example. We'll start with the sequence of integer cubes and below it write its successive differences.

$$\begin{aligned}
&a : 0, 1, 8, 27, 64, 125, 216, \ldots \\
&\Delta a : 1, 7, 19, 37, 61, 91, \ldots \\
&\Delta^2 a : 6, 12, 18, 24, 30, \ldots \\
&\Delta^3 a : 6, 6, 6, 6, \ldots \\
&\Delta^4 a : 0, 0, 0, \ldots
\end{aligned}$$

The obvious thing to notice here is that the fourth difference is identically zero. This is a general fact.

Theorem 6. If $f(0), f(1), \ldots$ is the sequence of values of a polynomial of degree d, then its first difference $\Delta f(i) = g(i)$ is given by a polynomial g of degree less than d, or is zero if $d = 0$.

Proof. If $d = 0$ then f is constant, so the first differences are clearly zero. If $f(x) = x^d$ then its first difference is given by $(x+1)^d - x^d$. Now $(x+1)^d$ is a polynomial of degree d with highest term x^d, so subtracting x^d leaves a polynomial of degree less than d. If $f(x)$ is a general polynomial of degree d, say

$$f(x) = c_d x^d + c_{d-1} x^{d-1} + \cdots + c_0$$

then

$$\Delta f(x) = c_d \Delta x^d + c_{d-1} \Delta x^{d-1} + \cdots \Delta c_0$$

By our calculation of Δx^k, this is a sum of polynomials of degree less than d and so is itself a polynomial of degree less than d.

Repeated application of the preceding theorem (or more precisely, mathematical induction) gives the following corollary.

Corollary 2. If $k > d$ then the k-th difference $\Delta^k f(i)$ of a polynomial f of degree d is identically zero.

This result provides a way of solving the key equation (2) in the decoding algorithm for $E(t)$, without having to compute $Q(t)$ at the same time.

Corollary 3. Let the received vector be

$$\begin{bmatrix} R_0 & R_1 & \ldots & R_{n-1} \end{bmatrix}$$

and let $a^{(j)}$ be the sequence

$$0^j R_0, 1^j R_1, \ldots, (n-1)^j R_{n-1}$$

Note that the $(m+e)$-th difference $\Delta^{m+e} a^{(j)}$ is a sequence of length $n - (m+e) = e$. Let $\mathbf{B}$ be the $e \times (e+1)$ matrix whose j-th column contains the entries

$$b_{ij} = \Delta^{m+e} a_i^{(j)}$$

of the sequence $\Delta^{m+e} a^{(j)}$, for $j = 0, 1, \ldots, e$. Then the coefficients v_k of the polynomial $E(t)$ in the decoding algorithm are given by the matrix equation

$$\mathbf{B} \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_e \end{bmatrix} = \mathbf{0}. \tag{3}$$

Proof. There isn't much to prove here, beyond unraveling what all the notation means. Since $Q(t)$ is a polynomial of degree $m + e - 1$ or less, the ($m + e$)-th difference of the sequence of values $Q(i)$ must be zero. Therefore the key equation gives

$$\Delta^{(m+e)} \left(R_i E(i) \right) = 0.$$

Writing this out in terms of the coefficients of the polynomial $E(t)$ leads directly to the matrix equation (3).

In most applications the error correction number e is much smaller than the code length n, so solving the system (3) of e linear equations in $e + 1$ unknowns is much quicker than solving the n equations in $n + 1$ unknowns resulting from the original key equation (2). However, we still have to find $Q(t)$ in order to compute $P(t)$. At this point we know the values $Q(0), \ldots, Q(n - 1)$, since the key equation gives

$$Q(i) = R_i E(i)$$

and we have found $E(t)$. So our remaining difficulty is to compute the coefficients of $Q(t)$ from its values. This ancient problem has a beautiful solution in terms of differences.

Theorem 7. Let $C_k(x)$ be the degree k polynomial

$$C_k(x) = x(x - 1) \cdots (x - k + 1)/k!$$

with $C_0(x)$ defined to be 1 . Let $f(x)$ be any polynomial of degree at most d. Then $f(x)$ can be computed from its successive differences by the formula

$$f(x) = f(0)C_0(x) + \Delta f(0)C_1(x) + \cdots + \Delta^d f(0)C_d(x). \tag{4}$$

Proof. We prove the theorem by mathematical induction on d. The base case $d = 0$ is trivial.

For the general case, since $\Delta f(x)$ has degree less than d we can use induction to apply formula (4) to $\Delta f(x)$, obtaining

$$\Delta f(x) = \Delta f(0)C_0(x) + \Delta^2 f(0)C_1(x) + \cdots + \Delta^{d+1} f(0)C_d(x)$$

Let $g(x)$ denote the expression on the right-hand side of formula (4). By direct calculation we can show that $\Delta C_k(x) = C_{k-1}(x)$ for all $k > 0$, and of course $\Delta C_0(x) = 0$. Using this we can calculate

$$\Delta g(x) = \Delta f(0)C_0(x) + \Delta^2 f(0)C_1(x) + \cdots + \Delta^{d+1} f(0)C_d(x)$$

which is equal to $\Delta f(x)$. In other words, $\Delta(f(x) - g(x)) = 0$, so $f(x) - g(x)$ is a constant. Now $C_k(0) = 0$ for all $k > 0$ and $C_0(0) = 1$, so $g(0) = f(0)$. This shows that the constant $f(x) - g(x)$ is equal to zero, proving the formula $f(x) = g(x)$.

Example. Looking back at the difference table for $f(x) = x^3$ in our earlier example, we see that $f(0) = 0, \Delta f(0) = 1, \Delta^2 f(0) = 6, \Delta^3 f(0) = 6$. According to the formula, we should have

$$x^3 = C_1(x) + 6C_2(x) + 6C_3(x),$$

which you can check is true by doing the algebra.

A bit of care is required when using formula (4) on polynomials with coefficients in $\mathbb{Z}_p$. In the definition of $C_k(x)$, we must understand the factor $1/k!$ to mean the inverse of $k!(\mathrm{mod} p)$. This inverse exists as long as $k < p$, and the formula works fine for polynomials of degree less than p. It does not work for higher degree polynomials. However, the polynomial $Q(t)$ we are trying to find in the decoding algorithm has degree $m + e - 1$, which is less than n, and $n \leq p$ by definition. Putting it all together, we get the improved algorithm.

Improved decoding algorithm. Given the received vector

$$\begin{bmatrix} R_0 & R_1 & \ldots & R_{n-1} \end{bmatrix},$$

compute the matrix $\mathbf{B}$ in Corollary 3, and find a non-zero solution of the resulting e linear equations for the $e + 1$ unknown coefficients v_j of $E(t)$. Compute the sequence of values $Q(i) = R_i E(i)$ and its difference table to find $Q(0), \Delta Q(0), \ldots, \Delta^{m+e-1} Q(0)$. Finally use formula (4) to compute $Q(t)$, and divide by $E(t)$ as before to find $P(t)$.

Example. We'll use a Reed-Solomon code over $\mathbb{Z}_7$ with $m = 3$ message symbols and $n = 7$ code symbols to correct $e = 2$ errors. Say the message vector is

$$\begin{bmatrix} 2 & 3 & 4 \end{bmatrix}.$$

The message polynomial is then

$$P(t) = 4t^2 + 3t + 2$$

and the code vector $\begin{bmatrix} P(0) & P(1) & \ldots & P(6) \end{bmatrix}$ is

$$\begin{bmatrix} 2 & 2 & 3 & 5 & 1 & 5 & 3 \end{bmatrix}.$$

All the arithmetic here is (mod 7), of course. Suppose we receive the vector with two errors as

$$\left[\begin{array}{ccccccc} 2 & 2 & 6 & 5 & 3 & 5 & 3 \end{array}\right].$$

This is our vector $\left[\begin{array}{cccc} R_0 & R_1 & \ldots & R_6 \end{array}\right]$. Following the recipe in Corollary 3, with $m+e=5$, we compute the fifth difference sequences

$$\begin{aligned} \Delta^5 R_i &= \left[\begin{array}{cc} 2 & 0 \end{array}\right] \\ \Delta^5 i R_i &= \left[\begin{array}{cc} 5 & 5 \end{array}\right] \\ \Delta^5 i^2 R_i &= \left[\begin{array}{cc} 0 & 2 \end{array}\right], \end{aligned}$$

yielding the matrix equation

$$\left[\begin{array}{ccc} 2 & 5 & 0 \\ 0 & 5 & 2 \end{array}\right] \left[\begin{array}{c} v_0 \\ v_1 \\ v_2 \end{array}\right] = \mathbf{0}$$

A nonzero solution (mod7) is

$$\left[\begin{array}{c} v_0 \\ v_1 \\ v_2 \end{array}\right] = \left[\begin{array}{c} 1 \\ 1 \\ 1 \end{array}\right],$$

giving $E(t) = t^2 + t + 1$. Now we compute the sequence $Q(i) = R_i E(i)$ to be

$$2, 6, 0, 2, 0, 1, 3$$

Its difference table is

$$\begin{array}{ccccccc} 2 & 6 & 0 & 2 & 0 & 1 & 3 \\ 4 & 1 & 2 & 5 & 1 & 2 & \\ 4 & 1 & 3 & 3 & 1 & & \\ 4 & 2 & 0 & 5 & & & \\ 5 & 5 & 5. & & & & \end{array}$$

Note that the next row would be the fifth difference, which is automatically zero because that was the condition we imposed to solve for $E(t)$. From the first column of the difference table and formula (4), we get

$$Q(t) = 2C_0(t) + 4C_1(t) + 4C_2(t) + 4C_3(t) + 5C_4(t)$$

and expanding out each $C_k(t)(\mathrm{mod}7)$, we find that this is equal to

$$Q(t) = 4t^4 + 2t^2 + 5t + 2 = \left(4t^2 + 3t + 2\right)\left(t^2 + t + 1\right)$$

As expected, the polynomial $E(t) = t^2 + t + 1$ factors out exactly, and the quotient $Q(t)/E(t) = 4t^2 + 3t + 2$ is the original message polynomial $P(t)$, recovered correctly despite the two errors.

The most complicated part of this whole procedure is the computation of the polynomials $C_k(t)$. Observe, however, that if we intended to decode many messages with the same code, we could compute these polynomials once and for all in advance, and quickly look up the result each time.

6. Can you patent a mathematical algorithm?

Historically, U.S. courts held that "laws of nature, natural phenomena, and abstract ideas" could not be patented. Mathematical algorithms were considered to be abstract ideas and therefore unpatentable. A computer program is a form of mathematical algorithm, and so, although the code for an individual program could be protected by copyright, the underlying algorithm could not be patented.

In the 1980's and 90's, patent attorneys for software companies began to file for patents on inventions consisting in part or entirely of algorithms, in a gradual effort to get the patent office and the courts to expand the definition of patentable inventions until it would include pure algorithms. In one famous case, ATT in 1988 patented an algorithm for linear programming developed by N. Karmarkar, a researcher at its Bell laboratories. Karmarkar's algorithm was an important mathematical development and the effort to patent it created much controversy, particularly because before the patent was applied for, Karmarkar and ATT boasted about how well the algorithm worked while refusing to reveal any details. The earlier Berlekamp-Welch patent seems to have attracted less attention.

When these patents were granted in the 80 's, their validity was doubtful, as courts then still held mathematical algorithms per se to be unpatentable. However, the broadening of the definition has continued to the point that under U.S. law today, there are effectively no barriers to patenting
software and algorithms. If an algorithm has "some type of practical application," the courts no longer consider it an unpatentable abstract idea.

In my opinion and that of many other mathematicians, and also software developers, the present state of affairs is a bad one. In mathematics, claims of ownership over ideas serve no useful purpose and only impede progress. In commerce and industry, patents are supposed to serve the public interest by providing a reward for innovation. However, many observers have made the case that software patents have the opposite overall effect, stifling innovation by threatening in effect to make much non-commercial software development illegal.

Patents and copyrights are often spoken of today as intellectual property. This rather loaded term carries with it the suggestion that it is natural or desirable for individuals or corporations to own ideas and control their dissemination and use, prohibiting their sharing in the public domain. Historically, however, the basis of patent law is not the concept of natural property but that of a social contract, in which society grants to inventors a temporary monopoly on the use of their inventions in return for publicizing those inventions and as an incentive to invent. It's interesting to note that the basis for the authority of Congress to establish patents and copyrights is the following clause in Article I, Section 8 of the U.S. Constitution, which explicitly specifies the temporary nature of such rights and the social

interest that they should serve:

The Congress shall have the power. . . To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries;

When patents on algorithms, software, genes, and life forms begin to obstruct more than to promote the "Progress of Science and useful Arts," perhaps the rules governing them need reconsideration.

Here are some sources of additional information and opinions on this subject

http://lpf.ai.mit.edu
(League for Programming Freedom web site)

www.softwarepatents.co.uk
(UK Resource on Software Patents: web site about UK and European software patents also discusses the U.S. experience)

http://swpat.ffii.org/index.en.html
(Foundation for a Free Information Infrastructure: German web site about European software patents)

7. Exercises

1. Consider $F(t) = 2t^5 + t^3 + 8t^2 + 1$ and $G(t) = t^2 + 3$ as polynomials with coefficients in $\mathbb{Z}_{11}$. Use long division to find a quotient $Q(t)$ and remainder $R(t)$ such that $F(t) = Q(t)G(t) + R(t)$ and $R(t)$ has smaller degree than $G(t)$.

2. Show that the polynomials $C_k(x)$ in Theorem 7 are uniquely determined by the properties (i) $C_k(x)$ has degree k, (ii) $C_k(i) = 0$ for $i = 0, 1, \ldots, k-1$, (iii) $C_k(k) = 1$.

3. Use the properties of $C_k(x)$ in Exercise 2 to show that $\Delta C_k(x) = C_{k-1}(x)$ for $k > 0$.

4. Show that a polynomial $f(x)$ has integer values $f(n)$ for every integer n if and only if it can be written as

$$f(x) = a_0 C_0(x) + a_1 C_1(x) + \cdots + a_d C_d(x)$$

for some d and integer coefficients $a_0, a_1, \ldots, a_d$.

5. Show that for integers $n \geq 0$, the value $C_k(n)$ is equal to the binomial coefficient $\binom{n}{k}$.

6. If you know that $1, 1, 2, 3, 3$ is the beginning of the sequence of values $f(0), f(1), f(2), \ldots$ of a polynomial of degree 3, find the next four terms in the sequence. [Hint: make a difference table, then extend each row four more terms, starting at the bottom.]

7. Find the polynomial $f(x)$ that produces the sequence in Exercise 6.

8. The Hamming code of order k is a linear code over $\mathbb{Z}_2$ with $n = 2^k - 1$ code symbols and $m = 2^k - k - 1$ message symbols, given by the code matrix

$$\mathbf{C} = [\mathbf{I}_m \mid \mathbf{B}],$$

where $\mathbf{B}$ is an $m \times k$ matrix whose rows are all the possible row vectors of length k with entries 0 and 1 , and at least two 1's (note that the number of possible vectors is equal to m).

(a) Write out the code matrix for a Hamming code of order 3.

(b) Show that the Hamming code of order k has weight 3, and therefore can correct one error. [Hint: this amounts to showing that every non-zero vector obtained as a sum of rows in $\mathbf{C}$ has at least three 1's in it.]

9. Show that the Hamming codes in Exercise 8 are perfect single-error correcting codes: every vector is either a code vector or has Hamming distance 1 from a code vector. This implies that no single-error correcting code over $\mathbb{Z}_2$ of the same length n can have more message symbols than the Hamming code.

10. Show that a Reed-Solomon code with 1 message symbol and n code symbols is just an n-fold redundancy code.

11. (a) Write out the code matrix $\mathbf{C}$ for the two-error correcting Reed-Solomon code over $\mathbb{Z}_7$ with 7 code symbols (the code used in the example in Section 5).

(b) Use the code matrix to encode the same message vector $\begin{bmatrix} 2 & 3 & 4 \end{bmatrix}$ as in the example, and check that your answer agrees with the one there.

12. For a linear code with $m \times n$ code matrix $\mathbf{C} = [\mathbf{I}_m \mid \mathbf{P}]$, show that a received vector $\mathbf{r}$ is a code vector (i.e., has no errors) if and only if $\mathbf{rS} = \mathbf{0}$, where

$$\mathbf{S} = \left[\begin{array}{c} \mathbf{P} \\ \hline -\mathbf{I}_{n-m} \end{array}\right]$$

13. Using a single-error correcting Reed-Solomon code over $\mathbb{Z}_{11}$ with 3 message symbols and 5 code symbols, you receive the vector $\begin{bmatrix} 9 & 2 & 9 & 1 & 7 \end{bmatrix}$. Where is the error and what were the transmitted code vector and the original message polynomial?

14. The data rate of an error correcting code is the fraction (number of message symbols)/(number of code symbols). The error rate of a transmission channel is the fraction of the code symbols transmitted in error. What is the maximum data rate achievable by a ReedSolomon code if it must be able to tolerate a maximum error rate of 1/3 ? How is this better than a triple-redundancy code?