Organization of the course

Flipped Classroom & Team-Based Learning

This course will be taught in flipped-classroom style with elements of team-based learning. For the duration of the whole course you will be divided into teams of five students, see below for details.

Check this video (in English) or this infographic (in Dutch) to learn more about flipped-classroom style. For you as students, the biggest difference to conventional lectures is that you have to **prepare yourself before attending classes**, as it is assumed during classes that you have studied the material already. The advantage of this model is that we can spend the face-to-face time during the work sessions to talk about the studied material together, to recapitulate the most difficult and important aspects, and to strengthen our understanding by actually working on the relevant problems in teams. The work sessions will be led by teacher and TAs, with an active involvement of the students.

Our week in Information Theory:

This is a 6 EC course given over 7 weeks, plus one week of exam. We are aiming to entertain you for approximately 20 hours per week with this course.

The content of every week is distributed via the following sources:

- 7 Canvas modules consisting of Pages and Quizzes, check out the overview and see the details under Modules
- 7 Weekly sets of (usually 6) practice problems. These problems will be solved during the work sessions, and presented during presentation sessions on Fridays, see below for details.
- 6 weekly homework sets, available from on Wednesday afternoon, to be handed in one week later on Wednesday at 12:00.

The weekly schedule looks as follows:

- Monday/Tuesday: individually study this week's Canvas modules, individually answer the reading questions (quizzes). There are two hours scheduled for

"self-study" in your schedules on Tuesday 11-13. No actual session will take place during this time, and you can study the material whenever you like.

- Wednesday 12:00: deadline for Reading Questions (individual) and Homework (team).
- Wednesday 14:00-17:00 (or 15:00-18:00, depending on your team): Work session, at 16:00 (or 17:00) new homework becomes available.
- Friday 12:00: presentation schedule available, 13:00-15:00: Presentation session.

Every week, your actions will be the following:

1. go through the material of the week
2. read up on the material in other sources from the internet
3. (individually) answer the reading questions (due at 12:00 **before** the work session)
4. during the work sessions, we start with an intro and team quiz as warmup (see below) and then work together in teams on practice problems. The goal is that every student is able to solve all of the practice problems. The work sessions are the chance to meet up and work with your team, so your attendance is expected. See below for details.
5. (in teams) carry out and hand in the homework problems (due 6 days after each work session).
6. (in teams) attend the presentation session, where teams present the practice problems.
7. if you have time and interest, check out the bonus material.

Work Session (Wed, 14:00-17:00 or 15:00-18:00)

Bring your own device**! Bring along a charged laptop or smartphone on which you can solve a Canvas quiz in order to complete the Intro Quiz.** If you have trouble accessing Canvas with your laptop, please go to the laptop helpdesk (Tue and Thu 12-13 next to the library desk).

We expect regular work sessions to proceed as follows:

- 14:00; start
- 14:05-14:25; Intro Quiz (individual quiz)

- 14:25-14:45; Team Quiz (group quiz), Team Quiz is to be submitted by at least one team member. The whole team will receive the grades of the team member who submits first. Team Quiz has the same questions as the Intro Quiz.
- 14:45- ~15:00; discussion about the Intro/Team Quiz, additional explanations of the week's material, previous homework questions etc.
- until 16:00; work on the problem sets, in your teams. Remember, the goal is that every student is able to solve all practice problems. Each team will be told which two (of the usually 6) problems will be the ones they specialize in one which they will present on Friday. As a team, start with the two problems you are assigned, work out and write down a clean solution on paper or electronically. Think about how you want to present this question on Friday. Get the attention of one of the available teachers and get this written version approved by the teacher. Once it has been approved, start working on the other problems. Prepare yourself to a degree that your team is able to follow the presentations of any of the other problems.
- 16:00; new homework is made available
- 16:00-16:45; distribute and plan your further team work on the problem sets and homework. Remember that at the final exam, all team members are expected to be able to solve all of the practice and homework problems individually.

Presentation Session (Friday 13:00-15:00)

Presentations sessions are held in groups of about 6 teams (so around 30 students):

SP G rooms: Some of these rooms might not have enough chairs, we will have to place some temporary chairs in them in order to fit everybody. If your presentation session is in this room, please help the teachers to set up the room and chairs, and also to remove them again afterwards.

A presentation schedule will be made available on Fridays at 12:00, it shows for each problem which team is is presenting it. For each problem, the presenting team is picked randomly among the teams who was preparing the problem.

The other team who was preparing the question (but is not presenting it) will be the designated moderators for this problem, i.e. your job is to ask clarifying questions, help whenever the presentation goes off path etc. The presentation of

a single problem is supposed to take around 10 minutes with another 5 minutes for questions and discussion, resulting in the available 6*15=90 minutes. However, presentation lengths might vary considerably with difficulty and quality of the presenter and moderator. It is entirely possible that some problems will not be presented due to time constraints.

Check our collection of useful tips both for moderators and presenters

Team Dynamics & Dropping the Course

You are working in the same fixed teams during the first three weeks. After three weeks, your performance will be evaluated by your fellow team members. **As courtesy to your team, we ask you to follow this course for at least the first three weeks.** That will be a natural point to drop the course if you don't see it fit. We will form new teams from those people who want to finish the course. These teams will then be fixed for the rest of the course. Everything works the same, just with different people.

Final Exam

There will be a written final exam probably on Wednesday, 18 Dec, 13:00-16:00. The exam will consist of two parts: a first quick multiple-choice part about concept questions, and then some problems to work out. The first hour (13:00 - 14:00) will be closed-book, i.e. WITHOUT notes/internet/etc, and at 14:00 you will be asked to hand in the first part of the exam (of course, you can already start working on the second part before 14:00, as soon as you are done with the first part). From 14:00-16:00, the exam is open-book, meaning that you can consult any study material including these Canvas pages, our lecture notes, [CF], [CT], [MacKay] as well as any notes you have made. You are allowed to use a laptop (or iPad, e-reader etc.) to access the course pages and websites like wikipedia, wolfram alpha etc. The exam is not an exercise in googling solutions, therefore we ask you not to use any solution manuals for information theory exercises and similar material that can be found online.

Obviously, you are also not allowed to communicate with each other nor with anybody else on the internet. Therefore, your computer screen has to be large enough (no smartphones) and visible during all the time you are using it. We are trusting you that you play according to these rules.

Exams from previous years are available for practice: Exam 2014, Exam 2015 (in 2014/2015 it was open-book, but no electronic devices were allowed), Exam 2017 (same style as this year).

The grade obtained at the exam counts for 50% towards the final grade, you have to have at least a score of 50% at the final exam to pass the course!

Grades

For all team assignments such as homework and the Team Quiz every week, all members of a team receive the same grade.

Your final grades consist of three parts:

- 15% is determined by the average grade for the reading questions, intro and team quizzes, as well as the team-member evaluation.
- 35% is by the average (team) homework grade
- 50% by the (individual) final exam, but you have to have at least a score of 50% at the final exam to pass the whole course.

Questions

If at any point you have a question, remarks, feedback etc. it's likely that you are not the only one wondering about this issue. Please raise it in the General Discussion forum, so other students or the teachers can answer them for everybody. If you want to get in touch with us directly, please use the Canvas-internal messaging function to send a **message to all teachers**. Please do not use regular email! [Test]

Course content (overview, now with video!)

(Download the slides from the video to "click on the links")

For a detailed overview of course content and learning objectives, see Studiegids.

The content of this course is organized into seven modules (one for each week the course runs, except the last week in which you will make a final exam):

01 Probability Theory

This is an introductory module that you will prepare before the start of the first lecture. It allows you to check your knowledge about logarithms, probability theory, and programming.

02 Entropy

This module introduces some of the core concepts of information theory: entropy, conditional entropy, and mutual information. You will learn how to work with these concepts, and how to relate them to each other.

03 Source Coding / Data Compression

In this module, you will learn how to compress sources of data that are not uniformly random (e.g., files of English text). You will see some specific algorithms (in pseudo-code) and learn why they perform optimally.

04 Typical Sets and Encryption

The first half of this module continues with the topic of data compression, and introduces the concept of typical sets to aid in efficient (but lossy) compression. In the second half, you will learn how to perform perfectly (information-theoretically) secure encryption, and what that necessarily costs.

05 Random Processes

In this module, you will study various random processes (such as Markov Chains), and learn how to measure their randomness using the concept of "entropy rate". We will also consider random walks on graphs in this module.

06 Error Correction and Zero-Error Transmission

This module considers the problem of trying to send a message over a noisy channel, and limiting the probability that the message (information) is lost while doing so. You will explore how much information you can send over different types of channels if you want to eliminate the probability of loss completely.

07 Noisy Channel Coding

This model continues with the topic of module 06, but takes a more fundamental approach. We will finish by proving Shannon's famous source-channel coding theorem, which determines how much information can be sent over a noisy channel if some (arbitrarily small) probability of losing the message is allowed.

Tips for Moderators and Presenters

The following tips are meant for moderators and presenters at the presentation sessions on Fridays. See course organization for details.

Goal for moderators: every student who is present at the session understands how to solve the problem. In other words, transfer your understanding and knowledge of the problem to the audience "through" the presenter.

Tips for moderators:

- Besides preparing a written “perfect solution” during the work session, also think about the following questions: What is “the point” of the exercise? Are there different solutions possible? What are the possible obstacles that (other) students might encounter? Discuss the answers to these questions with the teacher during the work session as well.
- Slow down presenters if they are moving too fast.
- Try to ask clarifying questions on the way.
- Keep an eye on time (e.g. by stopping students early enough if you see that the presentation is not going in the right way).
- If the presenters are stuck, ask them the right questions to help them on their way, or give them small hints. Do not take over the presentation by giving away the full solution immediately.

Tips for presenters:

- When preparing the exercise, think about which things you want to write down on the board, and which things you only want to talk about.
- It is a good idea to quickly recap what the problem is (possibly writing the essential parts on the board), but do not waste a lot of time by copying the whole exercise.
- Use the board, write down what you are doing.
- Try to write in a readable way. Always speak out what you write, it helps to decipher handwriting.
- Don’t go too fast, make sure everybody can follow and has understood the steps so far.

- Even as speaker, ask whether the audience has understood and could follow your explanations. How many have had the same solution?
- Don’t be afraid of making pauses and asking questions.

References and Shannon's original papers

Shannon's originals

Most of the contents of this course are based on (a modern understanding of) the contents of the two following papers by Claude Shannon:

A mathematical theory of communication - the original paper by Claude Shannon, published in 1948 in *The Bell System Technical Journal.* Check its wikipedia entry.

Communication Theory of Secrecy Systems is a paper published in 1949 by Claude Shannon discussing cryptography from the viewpoint of information theory. Check its wikipedia entry.

More recent references

For more contemporary readings of these papers, have a look at:

- Course lecture notes (work-in-progress) by Yfke Dulek and Christian Schaffner, available onhttps://github.com/cschaffner/InformationTheory/raw/master/Script/InfTheory3.pdf
- Good reference: [CT] Thomas M. Cover, Joy A. Thomas. 'Elements of information theory':http://onlinelibrary.wiley.com/book/10.1002/0471200611, 2nd Edition. New York: Wiley-Interscience, 2006. ISBN 0-471-24195-4.
- Good reference: [MacKay] David J. C. MacKay. 'Information Theory, Inference, and Learning Algorithms':http://www.inference.phy.cam.ac.uk/mackay/itila/book.html. Cambridge University Press, 2003. ISBN 0-521-64298-1

Copyright and Acknowledgements

The blended-learning material in this course was created from April to December 2018 by Yfke Dulek and Christian Schaffner, with the help of Huub Rutjes (interactive graphs) and Krijn Boom (videos) from the FNWI blended learning team. We were supported by a FNWI blended-learning grant. Thank you for the support!

Some of the material in this course is heavily inspired from:

- Jeremy Orloff, and Jonathan Bloom. *18.05 Introduction to Probability and Statistics, Spring 2014*. (Massachusetts Institute of Technology: MIT OpenCourseWare), http://ocw.mit.edu (Accessed). License: Creative Commons BY-NC-SA

Thanks a lot!

Introduction to Module 01

In this module, you will get an overview of the knowledge and skills we expect from you at the start of the course. Depending on your background, you might need to brush up on your probability theory, or you might want to grasp some basic programming skills.

Every piece of content ends with a small quiz. These quizzes allow you to assess your knowledge of that particular piece of content. You may choose to read the content first, or go directly to the quiz if you think you already master the content. **Important:** you need to finish all quizzes in this module before the start of the first working session.

If you want to build a stronger foundation for the topics discussed in this module, you can have a look at these resources:

- The lecture notes for the Basic Probability: Theory course taught at UvA.
- Programming tutorials such as Learn Python or Codecademy's Python tutorial.
- any source you find yourself on the world wide web.

Logarithms

Throughout the course, we will be working heavily with the logarithm function. Familiarize yourself with its definition, and make sure you are able to manipulate expressions and solve equations involving logarithms.

Convention

We will write $\log(x)$ for the base-2 logarithm of x. That is, $\log x = \frac{\ln(x)}{\ln(2)}$.

In particular, you should know the following facts:

$$
\begin{aligned}
\log(1) &= 0. \text{For } 0 < x < 1 \text{ it holds that } \log(x) < 0, \text{ and for } x > 1, \text{ we have that } \log(x) > 0. \\
\log(2) &= 1. \\
\log(2^x) &= x. \\
2^{\log(x)} &= x. \\
\log(x \cdot y) &= \log(x) + \log(y). \\
\log\left(\frac{x}{y}\right) &= \log(x) - \log(y). \\
\log(x^y) &= y \cdot \log(x). \\
x^{\log(y)} &= y^{\log(x)}.
\end{aligned}
$$

- The logarithm function is only defined on $\mathbb{R}_+$.
- The logarithm function is strictly increasing: $\log(x) > \log(y)$ whenever $x > y$.
- $\log(x+y)$ and $\log(x-y)$ do not generally have a simpler form.

Probability Spaces and Events

For this course, we will only be concerned with discrete probabilities. This section formalizes some notions you should already be familiar with: probability spaces, events and probability distributions.

Definition: Probability space

A (discrete) probability space $(\Omega, \mathcal{F}, P)$ consists of a discrete, non-empty *sample space* Ω, an *event space* $\mathcal{F} \subseteq \mathcal{P}(\Omega)$ (where $\mathcal{P}(\Omega)$ is the powerset of Ω) and a *probability measure* P which is a function $P : \Omega \to \mathbb{R}_{\geq 0}$ that satisfies

$$\sum_{\omega \in \Omega} P(\omega) = 1.$$

The event space $\mathcal{F}$ is required to be non-empty and closed under intersection, union and complements. For convenience, we will most often assume that $\mathcal{F}$ equals the powerset $\mathcal{P}(\Omega)$ of Ω, i.e., it contains all possible subsets of events, and therefore fulfils the required properties.

Definition: Event

An event $\mathcal{A}$ is an element of the event space $\mathcal{F} \subseteq \mathcal{P}(\Omega)$, i.e., a subset $\mathcal{A}$ of the sample space Ω. Its probability is defined as

$$P[\mathcal{A}] := \sum_{\omega \in \mathcal{A}} P(\omega),$$

where by convention $P[\emptyset] = 0$.

As a notational convention, we write $P[\mathcal{A}, \mathcal{B}]$ for $P[\mathcal{A} \cap \mathcal{B}]$, and $P[\overline{\mathcal{A}}]$ for $P[\Omega \backslash \mathcal{A}]$. The following identities hold for arbitrary events $\mathcal{A}, \mathcal{B} \subseteq \Omega$ (try to prove them for yourself):

- $P[\overline{\mathcal{A}}] = 1 - P[\mathcal{A}]$
- $P[\mathcal{A} \cup \mathcal{B}] = P[\mathcal{A}] + P[\mathcal{B}] - P[\mathcal{A}, \mathcal{B}]$
- $P[\mathcal{A}] = P[\mathcal{A}, \mathcal{B}] + P[\mathcal{A}, \overline{\mathcal{B}}]$.

It is often useful to consider the probability of an event *given* that some other event happened:

Definition: Conditional probability

For events $\mathcal{A}$ and $\mathcal{B}$ with $P[\mathcal{A}] > 0$, the conditional probability of $\mathcal{B}$ given $\mathcal{A}$ is defined as

$$P[\mathcal{B}|\mathcal{A}] := \frac{P[\mathcal{A}, \mathcal{B}]}{P[\mathcal{A}]}\,.$$

Example: Fair die

We throw a six-sided fair die once, and consider the number that comes up. The sample space for this experiment is $\Omega = 1, 2, 3, 4, 5, 6$, with event space $\mathcal{F} = \mathcal{P}(\Omega)$ and probability measure $P[i] = \frac{1}{|\Omega|} = \frac{1}{6}$ for all $i \in \Omega$ (this is a **uniform** probability measure). Consider the events $\mathcal{A} = 2, 4, 6$ and $\mathcal{B} = 3, 6$. Using the formulas in the definitions of events and conditional probabilities, we can compute the following probabilities:

$$\begin{aligned}
P[\mathcal{A}] &= \frac{1}{2} (\text{the outcome is even}) \\
P[\mathcal{B}] &= \frac{1}{3} (\text{the outcome is a multiple of } 3) \\
P[\mathcal{A}, \mathcal{B}] = P[6] &= \frac{1}{6} (\text{the roll is even and a multiple of } 3) \\
P[\mathcal{A}|\mathcal{B}] = \frac{1/6}{1/3} &= \frac{1}{2} (\text{the roll is even, given that it is a multiple of } 3) \\
P[\mathcal{B}|\mathcal{A}] = \frac{1/6}{1/2} &= \frac{1}{3} (\text{the roll is a multiple of } 3, \text{ given that it is even})
\end{aligned}$$

This example shows that in general, $P[\mathcal{A}|\mathcal{B}]$ is *not necessarily equal* to $P[\mathcal{B}|\mathcal{A}]$. In fact, they are related through Bayes' rule.
decide whether this button is necessary, and whether the title of this block should be question instead of example

Random Variables and Distributions

Definition: Discrete Random Variable (RV)

Let $(\Omega, \mathcal{F}, P)$ be a discrete probability space. A random variable X is a function $X : \Omega \to \mathcal{X}$ where $\mathcal{X}$ is a set, and we may assume it to be discrete.

A *real* random variable is one whose image is contained in $\mathbb{R}$. A (The *image* and the *range* of a random variable X are given by the image and the range of X in the function-theoretic sense.) The image of a *binary* random variable is a set x_0, x_1 with only two elements.

Definition: Probability distribution

Let X be a random variable. The probability distribution of X is the function $P_X : \mathcal{X} \to [0, 1]$ defined as

$$P_X(x) := P[X = x],$$

where $X = x$ denotes the event $\{\omega \in \Omega \mid X(\omega) = x\}$.

Alternatively, one can write $P_X(x) = P[X^{-1}(x)]$ to express that the probability of x is precisely the P-measure of the pre-image of x under the random variable X.

We say that P_X is a **uniform** distribution if the associated probability measure is uniform, i.e. $P_X(x) = \frac{1}{|\mathcal{X}|}$. The **support** of a random variable or a probability distribution is defined as $\operatorname{supp}(P_X) := \{x \in \mathcal{X} \mid P_X(x) > 0\}$, the points of the range which have strictly positive probability. We often slightly abuse notation and write $\operatorname{supp}(X)$ instead. When given two or more random variables defined on the same probability space, we can consider the probability that each of the variables take on a certain value:

Definition: Joint probability distribution

Let X and Y be two random variables defined on the same probability space, with respective ranges $\mathcal{X}$ and $\mathcal{Y}$. The pair XY is a random variable with probability distribution $P_{XY} : \mathcal{X} \times \mathcal{Y} \to [0, 1]$ given by

$$P_{XY}(x, y) := P[X = x, Y = y].$$

This definition naturally extends to three and more random variables. Unless otherwise stated, a collection of random variables is assumed to be defined on the same (implicit) probability space, so that their joint distribution is always well-defined. If $P_{XY} = P_X \cdot P_Y$, in the sense that $P_{XY}(x, y) = P_X(x) P_Y(y)$ for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, then the random variables X and Y are said to be **independent**. If a set of variables $X_1, \ldots, X_n$ are all mutually independent and all have the same distribution (i.e., $P_{X_i} = P_{X_j}$ for all i, j), then they are **independent and identically distributed**, or **i.i.d.** From a joint distribution, we can

always find out the "original'' (or **marginal**) distribution of one of the random variables (for example, X) by **marginalizing** out the variable that we want to discard (for example, Y):

$$P_X(x) = \sum_{y \in \mathcal{Y}} P_{XY}(x, y).$$

This marginalization process also works with more than two random variables. Like events, probability distributions can also be conditioned on probabilistic events:

Definition: Conditional probability distribution

If $\mathcal{A}$ is an event with $P[\mathcal{A}] > 0$, then the conditional probability distribution of X given $\mathcal{A}$ is given by

$$P_{X|\mathcal{A}}(x) = \frac{P[X = x, \mathcal{A}]}{P[\mathcal{A}]}.$$

If Y is another random variable and $P_Y(y) > 0$, then we write

$$P_{X|Y}(x|y) := P_{X|Y=y}(x) = \frac{P_{XY}(x, y)}{P_Y(y)}$$

for the conditional distribution of X, given $Y = y$.

Note that again, both $(\mathcal{X}, P_{X|\mathcal{A}})$ and $(\mathcal{X}, P_{X|Y=y})$ themselves form probability spaces. Note also that if X and Y are independent, then

$$P_{X|Y}(x|y) = \frac{P_{XY}(x, y)}{P_Y(y)} = \frac{P_X(x) \cdot P_Y(y)}{P_Y(y)} = P_X(x),$$

which aligns well with our intuition of independent variables: the distribution of X remains unchanged when Y is fixed to a specific value.

Example: Fair die (continued)

Consider again the throw of a six-sided fair die. Let the random variable X describe the number of (distinct) integer divisors for the outcome, that is

$$X(1) = 1 \quad X(2) = 2 \quad X(3) = 2 \quad X(4) = 3 \quad X(5) = 2 \quad X(6) = 4$$

X is a real random variable, with range $\mathcal{X} = 1, 2, 3, 4$. The associated probability distribution is

$$P_X(1) = P[1] = \frac{1}{6}, \quad P_X(2) = P[2, 3, 5] = \frac{1}{2}, \quad P_X(3) = P[4] = \frac{1}{6}, \quad P_X(4) = P[6] = \frac{1}{6}.$$

If we now condition on the event $\mathcal{A} = 2, 4, 6$ (the outcome of the die being even), we get that

$$P_{X|\mathcal{A}}(1) = 0, \quad P_{X|\mathcal{A}}(2) = \frac{1}{3}, \quad P_{X|\mathcal{A}}(3) = \frac{1}{3}, \quad P_{X|\mathcal{A}}(4) = \frac{1}{3}$$

If X is a random variable and $f : \mathcal{X} \to \mathcal{Y}$ is a surjective function, then $f(X)$ is a random variable, defined by composing the map f with the map X. Its image is $\mathcal{Y}$. Clearly,

$$P_{f(X)}(y) = \sum_{x \in \mathcal{X}: f(x)=y} P_X(x).$$

For example, $1/P_X(X)$ denotes the real random variable obtained from another random variable X by composing with the map $1/P_X$ that assigns $1/P_X(x) \in \mathbb{R}$ to $x \in \mathcal{X}$.

Expectation and Variance

Definition: Expectation

The expectation of a *real* random variable X is defined as

$$\mathbb{E}[X] := \sum_{x \in \mathcal{X}} P_X(x) \cdot x.$$

Note that if X is not real, then we can still consider the expectation of some function $f : \mathcal{X} \to \mathbb{R}$, where

$$\mathbb{E}[f(X)] = \sum_{x \in \mathcal{X}} P_X(x) \cdot f(X).$$

Definition: Variance

The variance of a *real* random variable X is defined as

$$\mathrm{Var}[X] := \mathbb{E}[(X - \mathbb{E}[X])^2].$$

The variation is a measure for the deviation of the mean. Hoeffding's inequality (here stated for binary random variables) states that for a list of i.i.d. random variables, the average of the random variables is close to the expectation, except with very small probability. We state it here without proof.

Theorem: Hoeffding's inequality

Let $X_1, \ldots, X_n$ be independent and identically distributed binary random variables with $P_{X_i}(0) = 1 - \mu$ and $P_{X_i}(1) = \mu$, and thus $\mathbb{E}[X_i] = \mu$. Then, for any $\delta > 0$

$$P\left[\sum_i X_i > (\mu + \delta) \cdot n\right] \leq \exp(-2\delta^2 n).$$

Some Important Distributions

We end the theoretic preliminaries with a (non-exhaustive) list of common probability distributions that you will come across throughout the course.

- The distribution of a biased coin with probability $P_X(1) = p$ to land heads, and a probability of $P_X(0) = 1 - p$ to land tails is called **Bernoulli(p)** distribution. The expected value is $\mathbb{E}[X] = p$ and the variance is $\mathrm{Var}[X] = p(1-p)$.
- When n coins $X_1, X_2, \ldots, X_n$ are flipped independently and every X_i is Bernoulli(p) distributed, let $S = \sum_{i=1}^{n} X_i$ be their sum, i.e., the number of heads in n throws of a biased coin. Then, S has the **binomial(n, p)** distribution:

  $$P_S(k) = \binom{n}{k} p^k (1-p)^{n-k} \qquad \text{where } k = 0, 1, 2, \ldots, n\,.$$

  From simple properties of the expected value and variance, one can show that $\mathbb{E}[S] = np$ and $\mathrm{Var}[S] = np(1-p)$.
- The **geometric(p)** distribution of a random variable Y is defined as the number of times one has to flip a Bernoulli(p) coin before it lands heads:

  $$P_Y(k) = (1-p)^{k-1} p \qquad \text{where } k = 1, 2, 3, \ldots\,.$$

  There is another variant of the geometric distribution used in the literature, where one excludes the final success event of landing heads in the counting:

  $$P_Z(k) = (1-p)^{k} p \qquad \text{where } k = 0, 1, 2, 3, \ldots\,.$$

  While the expected values are slightly different, namely $\mathbb{E}[Y] = \frac{1}{p}$ and $\mathbb{E}[Z] = \frac{1-p}{p}$, their variances are the same: $\mathrm{Var}[Y] = \mathrm{Var}[Z] = \frac{1-p}{p^2}$.

created: 2019-10-21

What? Do I need to know how to program for this course?

This is a theoretical course on information theory. However, these days, even if you are studying theory, some basic programming can help you tremendously. In this course, you will be expected to write small programs for exactly that purpose: to better understand certain concepts; to develop or test hypotheses; and to observe the effects of large numbers (bigger data sets which you cannot process by hand) on information-theoretic quantities.

Do I need to be a proficient programmer to follow this course?

No. We intentionally keep the programming exercises small, a few lines of code at a time. You will never be expected to write complicated pieces of software, and the emphasis will lie on the mathematical concepts rather than algorithmic complexity.

If you know about the following programming concepts, then you are good to go:

- instantiating variables, and giving them values of basic types (boolean, integer, float, string)
- performing basic operations on those types, such as adding integers, converting strings to lowercase, etc. (or knowing where to look up how to do these things)
- printing output to the console
- if(/else) statements and for/while loops
- define and call functions
- working with some basic data structures, such as lists (arrays) and dictionaries (maps).
- if you're not using Python: read an input plaintext file, and convert it into a useful data structure. For example, you may want to create an array of all words in the file, or a list of all different symbols and their frequencies.

What if I don't know how to do these things?

There are all kinds of free tools available to learn the basics of programming. If you have no programming experience yet, we advise you to use Python, because that will allow you to use the exercise files that we provide. You can have a look at one of the following online tutorials (or find your own):

- Learn Python
- Codecademy

Additionally, you will likely have one or more **team mates** that have programming experience; they may be willing to help you get through the hard bits. You can also ask other students for help using this discussion thread.

Do you provide IT support?

No, we do not. You are free to use any language/tool you like, but you are responsible for getting that language/tool up and running. We test the Python code stubs in the online environment in which we provide them; you can always fall back on using that.

There are also plenty of online environments available which don't require you to install anything (except sometimes create an account). For example, Repl.it is what we use to provide you with the Python code stubs to get you started. You can fork and edit these stubs directly on that website. This type of website also exists for Java and C++, for example.

What programming languages am I allowed to use?

We do not put any restrictions on the programming languages that you use for this course. We do *not* test your programming skills and we do *not* require you to hand in any code files or documentation. In this course, as is the case for many fields of research these days, programming is a supportive tool. It helps you to grasp certain definitions/concepts better, and it allows you to perform larger calculations that you would not be able to do by hand.

It does not matter if you use Java, C, Haskell, Python, Ruby, Excel, Mathematica, Rust, LaTeX (hey, it's Turing complete), bash, or Piet. As long as you are able to process input files in plaintext format, and get numerical/textual output. The output is all that matters.

If you have little or no programming experience, we advise you to write your programs in Python. We will provide you with python code stubs (starter files) that take care of the not-so-interesting parts for you: reading input files, storing their contents in arrays or other data structures, etc. That allows you to focus on the mathematically interesting parts of the program you are writing.

Introduction to Module 02

Welcome to the second module of the course! This week, we will build important foundations for the rest of the course. In particular, we will see *how to measure information* using entropy. You will learn about various quantities (such as conditional entropy, or mutual information) that can express information content in various different scenarios, and you will practice with how to formalize "real-world" scenarios (such as throwing dice) using these quantities. All of these tools will come back throughout the other modules.

Before we introduce the concept of entropy (information content), we need a mathematical tool: Jensen's inequality.

As an addition to the written material in this module, you may find it useful to watch lecture recordings as well. A good source (which has a lot of overlap, but does not follow the contents of our course exactly!) is this lecture series by David MacKay.

Definition: Convexity and Concavity

In the following, let $\mathcal{D}$ be an interval in $\mathbb{R}$.

Definition: Convex and concave functions

The function $f : \mathcal{D} \to \mathbb{R}$ is convex if for all $x_1, x_2 \in \mathcal{D}$ and all $\lambda \in [0,1] \subset \mathbb{R}$:

$$\lambda f(x_1) + (1-\lambda) f(x_2) \geq f(\lambda x_1 + (1-\lambda) x_2).$$

The function f is *strictly* convex if equality only holds when $\lambda \in \{0,1\}$ or when $x_1 = x_2$. The function f is (strictly) concave if the function $-f$ is (strictly) convex.

Intuitively, a function is convex if any straight line drawn between two points $f(x_1)$ and $f(x_2)$ lies above the graph of f entirely. For a concave function, such a line must lie entirely beneath the graph. Play around with the interactive figure below to understand the formulas above! Note that you can move the red slider for λ, and you can also adjust x_0, x_1 directly in the graph.

Convex or concave? MacKay's Mnemonic

David MacKay has a great way of remembering "which way" convex and concave functions go, namely by noticing the following. When pronouncing the word "convex", one could continue to say "smile", while pronouncing the word "concave", it could be followed by the word "frown".

Video-2018-05-16-12-30-51_MacKays mnemonic.MP4

The following proposition establishes a formal method of proving the convexity of a function.

Proposition

Let $f : \mathcal{D} \to \mathbb{R}$. If $\mathcal{D}$ is open, and for all $x \in \mathcal{D}$, the second-order derivative $f''(x)$ exists and is non-negative (positive), then f is convex (strictly) convex.

Proof

We omit the proof, which can be found in, for example, [CF] (Lemma 1).

Jensen's Inequality

The following theorem will be very useful to derive basic properties of entropy.

Theorem: Jensen's inequality

Let $f : \mathcal{D} \to \mathbb{R}$ be a convex function, and let $n \in \mathbb{N}$. Then for any $p_1, \ldots, p_n \in \mathbb{R}_{\geq 0}$ such that $\sum_{i=1}^{n} p_i = 1$ and for any $x_1, \ldots, x_n \in \mathcal{D}$ it holds that

$$\sum_{i=1}^{n} p_i f(x_i) \geq f\left(\sum_{i=1}^{n} p_i x_i\right).$$

If f is strictly convex and $p_1, \ldots, p_n > 0$, then equality holds if and only if $x_1 = \cdots = x_n$. In particular, if X is a real random variable whose image $\mathcal{X}$ is contained in $\mathcal{D}$, then

$$\mathbb{E}[f(X)] \geq f(\mathbb{E}[X]),$$

and if f is strictly convex, equality holds if and only if there is a $c \in \mathcal{X}$ such that $X = c$ with probability 1.

Proof

The proof is by induction. The case $n = 1$ is trivial, and the case $n = 2$ is identical to the very definition of convexity. Suppose that we have already proved the claim up to $n - 1 \geq 2$. Assume, without loss of generality, that $p_n < 1$. Then:

$$\begin{aligned}
\sum_{i=1}^{n} p_i f(x_i) &= p_n f(x_n) + \sum_{i=1}^{n-1} p_i f(x_i) \\
&= p_n f(x_n) + (1 - p_n) \sum_{i=1}^{n-1} \frac{p_i}{1 - p_n} f(x_i) \\
&\geq p_n f(x_n) + (1 - p_n) f\left(\sum_{i=1}^{n-1} \frac{p_i}{1 - p_n} x_i\right) \text{(induction hypothesis)} \\
&\geq f\left(p_n x_n + (1 - p_n) \sum_{i=1}^{n-1} \frac{p_i}{1 - p_n} x_i\right) \text{(definition of convexity)} \\
&= f\left(p_n x_n + \sum_{i=1}^{n-1} p_i x_i\right) \\
&= f\left(\sum_{i=1}^{n} p_i x_i\right).
\end{aligned}$$

That proves the claim. As for the strictness claim, if $x_1, \ldots, x_n$ are not all identical, then either $x_1, \ldots, x_{n-1}$ are not all identical and the first inequality is strict by induction hypothesis, or $x_1 = \cdots = x_{n-1} \neq x_n$ so that the second inequality is strict by the definition of convexity.

Definition: Shannon Entropy

In this section, we explore a measure for the amount of uncertainty of random variables. Watch the video below for an introduction to this topic:

Consider some probabilistic event $\mathcal{A}$ that occurs with probability $P[\mathcal{A}]$ for some probability measure P. The **surprisal value** $\log \frac{1}{P[\mathcal{A}]}$ indicates how surprised we should be when the event $\mathcal{A}$ occurs: events with small probabilities yield high surprisal values, and vice versa. An event that occurs with certainty ($P_X(\mathcal{A}) = 1$) yields a surprisal value of 0.

For a random variable X, we consider the *expected* surprisal value to be an indicator of how much uncertainty is contained in the variable, or how much information is gained by revealing the outcome. This expected surprisal value is more commonly known as the (Shannon) entropy of a random variable:

Definition: Entropy

Let X be a random variable with image $\mathcal{X}$. The (Shannon) entropy $H(X)$ of X is defined as

$$H(X) := \sum_{x \in \mathcal{X}} P_X(x) \cdot \log \frac{1}{P_X(x)} = -\sum_{x \in \mathcal{X}} P_X(x) \cdot \log P_X(x)\,,$$

with the convention that the $\log$ function represents the *binary* logarithm $\log_2$.

There are three things to note about this definition:

- The entropy of X is a function (solely) of the *distribution* P_X of X, and the elements of $\mathcal{X}$ (i.e., the values that X can take on) are completely irrelevant for the entropy. Therefore, it would be formally more correct to write $H(P_X)$. However, it is customary (and more convenient) to write $H(X)$ whenever there is no ambiguity about the underlying distribution.
- The summand $P_X(x) \cdot \log P_X(x)$ is technically undefined whenever $P_X(x) = 0$. As a convention, we set $P_X(x) \cdot \log P_X(x) = 0$ in this case. This choice is justified by taking the limit where $P_X(x)$ goes to 0 (see the exercise below).
- The entropy of X can also be expressed as the expectation of the random variable $\log\big(1/P_X(X)\big)$:

$$H(X) = \mathbb{E}_X\left[\log \frac{1}{P_X(X)}\right].$$

Exercise

Prove that $\lim_{p\to 0^+} p\log(p) = 0$. This is a justification for the convention that we set $P_X(x)\cdot \log P_X(x) = 0$ whenever $P_X(x) = 0$.

Show solution

First, note that

$$\lim_{p\to 0^+} \log p = -\infty$$

and

$$\lim_{p\to 0^+} \frac{1}{p} = \infty.$$

So, by l'Hopital's rule,

$$\lim_{p\to 0^+} p\log p = \lim_{p\to 0^+} \frac{\log p}{1/p} = \lim_{p\to 0^+} \frac{[\log p]'}{[1/p]'} = \lim_{p\to 0^+} -\ln(2)\frac{1/p}{1/p^2} = \lim_{p\to 0^+} -\ln(2)p = 0.$$

Example

Consider a random variable X with $\mathcal{X} = a, b, c, d$ and $P_X(a) = \frac{1}{2}$, $P_X(b) = P_X(c) = \frac{1}{4}$, $P_X(d) = 0$. The entropy of X is

$$H(X) = \frac{1}{2}\log 2 + \frac{1}{4}\log 4 + \frac{1}{4}\log 4 + 0 = \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + 0 = \frac{3}{2}.$$

Definition: Binary Entropy

For a binary random variable X with image $\mathcal{X} = \{x_0, x_1\}$ and probabilities $P_X(x_0) = p$ and $P_X(x_1) = 1 - p$, we can write $H(X) = h(p)$, where h denotes the binary entropy function:

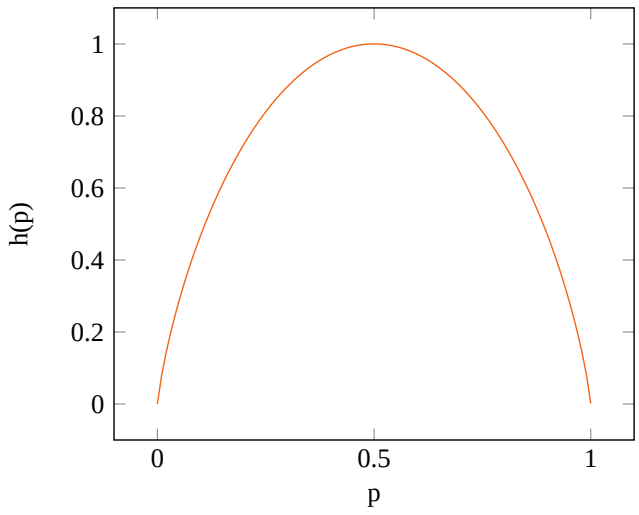
Definition: Binary entropy function h

The binary entropy function is defined for $0 < p < 1$ as

$$h(p) := p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p},$$

and is defined as $h(p) = 0$ for $p = 0$ or $p = 1$.

The graph of h on the interval $[0, 1]$, as a function of p, looks as follows:



If we think of X as the random variable describing the outcome of a coin flip, we see that a relatively fair coin ($p \approx \frac{1}{2}$) yields a higher expected surprisal value than a very biased coin (where p is closer to 0 or 1). If the coin is completely fair ($p = \frac{1}{2}$), the entropy is exactly 1 bit.

Properties of Shannon Entropy

In this section, we list a few properties of the Shannon entropy, and show a trick to compute it more easily by hand.

Proposition: Positivity of entropy

Let X be a random variable with image $\mathcal{X}$. Then

$$0 \leq H(X).$$

Equality holds iff there exists $x \in \mathcal{X}$ with $P_X(x) = 1$ (and thus $P_X(x') = 0$ for all $x' \neq x$).

Proof

For all $x \in \{X\}$, we have $0 \leq P_X(x) \leq 1$, and hence $-P_X(x) \log P_X(x) \geq 0$. So $H(X)$, which is the sum of those terms, is always nonnegative. To characterize the condition for equality, note that by definition of Shannon entropy, $H(X) = 0$ when $P_X(x) = 1$ for some x. On the other hand, if $H(X) = 0$ then for any x with $P_X(x) > 0$ it must be that $\log(1/P_X(x)) = 0$ and hence $P_X(x) = 1$.

Proposition: Upper bound on entropy

Let X be a random variable with image $\mathcal{X}$. Then

$$H(X) \leq \log(|\mathcal{X}|).$$

Equality holds iff $P_X(x) = 1/|\mathcal{X}|$ for all $x \in \mathcal{X}$.

Proof hint

We encourage you to try to find the proof for this proposition yourself. As a first step, you may want to write out the definition of $H(X)$ apply Jensen's inequality.

Show full proof

The function $f : \mathbb{R}_{>0} \to \mathbb{R}$ defined by $y \mapsto \log y$ is strictly concave on $\mathbb{R}_{>0}$. Thus, by Jensen's inequality:

$$H(X) = \sum_{x \in \mathcal{X}} P_X(x) \cdot \log \frac{1}{P_X(x)} \leq \log\Big(\sum_{x \in \mathcal{X}} P_X(x) \cdot \frac{1}{P_X(x)}\Big) = \log\Big(\sum_{x \in \mathcal{X}} 1\Big) = \log(|\mathcal{X}|).$$

Furthermore, since we may restrict the sum to all x with $P_X(x) > 0$, equality holds if and only if $\log(1/P_X(x)) = \log(1/P_X(x'))$, and thus $P_X(x) = P_X(x')$, for all $x, x' \in \mathcal{X}$.

When working with explicit distributions, one can always compute the entropy of a random variable by filling in all the probabilities in the definition of entropy. However, in some cases there is some structure to the distribution. In those cases, the entropy can be computed in a smarter and faster way. This is especially useful when you are computing the entropy by hand, but can also help when analysing the entropy of a more complex distribution containing some unknown variables.

Video-2018-06-27-12-10-50_Smart ways to compute entropy.MP4

In general, the entropy of a random variable with probabilities $p_1, \ldots, p_n$ can be expressed as

$$H(p_1, \ldots, p_k, p_{k+1}, \ldots, p_n) = h\left(\sum_{i=1}^{k} p_i\right) + \left(\sum_{i=1}^{k} p_i\right) \cdot H\left(\frac{p_1}{\sum_{i=1}^{k} p_i}, \ldots, \frac{p_k}{\sum_{i=1}^{k} p_i}\right) + \left(\sum_{i=k+1}^{n} p_i\right) \cdot H\left(\frac{p_k}{\sum_{i=k}^{n}}\right.$$

You can of course use this trick multiple times in a row to break down the entropies on the right-hand side of this equation even further.

Exercise

Consider a random variable X with $\mathcal{X} = a, b, c$ and $P_X(a) = \frac{1}{2}$, $P_X(b) = P_X(c) = \frac{1}{4}$. Compute the entropy of X using the techniques shown in the video.

Show solution

We can think of this distribution as the result of two fair coin tosses: if the first coin comes out heads, the outcome is a. If it comes out tails, we toss another fair coin to determine whether the outcome is b or c.

An appropriate underlying probability space (Ω, P) could be $\Omega = \mathsf{hh}, \mathsf{ht}, \mathsf{th}, \mathsf{tt}$ and $P(\omega) = \frac{1}{4}$ for all $\omega \in \Omega$. Then we define the function $X : \Omega \to \mathcal{X}$ as

$$X(\mathsf{hh}) = X(\mathsf{ht}) = a, \qquad X(\mathsf{th}) = b, \qquad X(\mathsf{tt}) = c.$$

This yields the correct distribution P_X.

The following computation now leads to the entropy of X:

$$H(X) = h\left(\frac{1}{2}\right) + \frac{1}{2}h(0) + \frac{1}{2}h\left(\frac{1}{2}\right) = \frac{3}{2}.$$

The first coin toss determines whether the outcome is a (on heads h) or something else (on tails t). On heads, the second coin toss does not give any more information, whereas on tails, the second coin toss still decides between outcome b and outcome c.

Definition: Conditional Entropy

Let X be a random variable and $\mathcal{A}$ an event. Applying the definition of entropy to the conditional probability distribution $P_{X|\mathcal{A}}$ allows us to naturally define the entropy of X conditioned on the event $\mathcal{A}$:

$$H(X|\mathcal{A}) := \sum_{x \in \mathcal{X}} P_{X|\mathcal{A}}(x) \cdot \log \frac{1}{P_{X|\mathcal{A}}(x)}.$$

This leads to the following notion:

Definition: Conditional entropy

Let X and Y be random variables, with respective images $\mathcal{X}$ and $\mathcal{Y}$. The conditional entropy $H(X|Y)$ of X given Y is defined as

$$H(X|Y) := \sum_{y \in \mathcal{Y}} P_Y(y) \cdot H(X|Y{=}y)\,,$$

with the convention that the corresponding argument in the summation is 0 for $y \in \mathcal{Y}$ with $P_Y(y) = 0$.

Note that the conditional entropy $H(X|Y)$ is not the entropy of a probability distribution but an expectation: the average uncertainty about X when given Y.

created: 2019-10-21

Bounds on the Conditional Entropy

On this page, we show that the conditional entropy $H(X|Y)$ is lower bounded by zero, and upper bounded by the entropy $H(X)$. That is, *on average* any additional information (i.e., knowing Y) can only *decrease* the uncertainty about X.

Proposition

Let X and Y be random variables with respective images $\mathcal{X}$ and $\mathcal{Y}$. Then

$$0 \leq H(X|Y) \leq H(X)$$

Equality on the left-hand side holds iff X is determined by Y, i.e., for all $y \in \mathcal{Y}$, there is an $x \in \mathcal{X}$ such that $P_{X|Y}(x|y) = 1$. Equality on the right-hand side holds iff X and Y are independent.

Proof

The lower bound follows trivially from the definition and from the positivity of entropy, and so does the characterization of when $H(X|Y) = 0$.

For the upper bound, note that

$$H(X|Y) = \sum_y P_Y(y) \sum_x P_{X|Y}(x|y) \log \frac{1}{P_{X|Y}(x|y)} = \sum_{x,y} P_{XY}(x,y) \log \frac{P_Y(y)}{P_{XY}(x,y)}$$

and

$$H(X) = \sum_x P_X(x) \log \frac{1}{P_X(x)} = \sum_{x,y} P_{XY}(x,y) \log \frac{1}{P_X(x)}$$

where the last equality is derived by marginalization. In both expressions, we may restrict the sum to those pairs (x,y) with $P_{XY}(x,y) > 0$.

Using Jensen's inequality, it follows that

$$\begin{aligned} H(X|Y) - H(X) &= \sum_{x,y:P_{XY}(x,y)>0} P_{XY}(x,y) \log \frac{P_X(x)P_Y(y)}{P_{XY}(x,y)} \\ &\leq \log\Big(\sum_{x,y:P_{XY}(x,y)>0} P_X(x)P_Y(y) \Big) \\ &\leq \log\Big(\sum_{x,y} P_X(x)P_Y(y) \Big) \\ &= \log\Big(\big(\sum_{x\in\mathcal{X}} P_X(x)\big)\big(\sum_{y\in\mathcal{Y}} P_Y(y)\big) \Big) \\ &= \log 1 = 0\,. \end{aligned}$$

The second inequality follows by the monotonicity of the logarithm function.

It remains to show that these inequalities are equalities if and only if X and Y are independent. Try it yourself first; you can use the equality condition of Jensen's inequality to characterize the first inequality in the derivation above.

Show solution

We start with the if-direction. Suppose that X and Y are independent, i.e., $P_X(x)P_Y(y) = P_{XY}(x,y)$ for all $(x,y) \in \mathcal{X}\times\mathcal{Y}$. Then for all $(x,y),(x',y') \in \mathcal{X}\times\mathcal{Y}$,

$$\frac{P_X(x)P_Y(y)}{P_{XY}(x,y)} = 1 = \frac{P_X(x')P_Y(y')}{P_{XY}(x',y')}.$$

By the equality condition in Jensen's inequality, the first inequality in the derivation above is an equality. The second inequality in the derivation above is easily seen to be equality as well: the second term is bigger only because we add those summands $P_X(x)P_Y(y)$ for which $P_{XY}(x,y) = 0$, but for those x and y, $P_X(x)P_Y(y) = 0$ as well. For the only-if-direction, suppose that $H(X|Y) = H(X)$, that is, both inequalities in the above derivation are equalities. Then (from the fact that the second inequality is an equality), we know that whenever $P_{XY}(x,y) = 0$, also $P_X(x)P_Y(y) = 0$. When $P_{XY}(x,y) > 0$, we know from the equality condition in Jensen's inequality that for all $y' \in \mathcal{Y}$ for which $P_{XY}(x,y') > 0$,

$$\frac{P_X(x)P_Y(y)}{P_{XY}(x,y)} = \frac{P_X(x)P_Y(y')}{P_{XY}(x,y')}.$$

Working out the term $P_{XY}(x,y)$ as $P_{X|Y}(x|y)P_Y(y)$ (and similarly for $P_{XY}(x,y')$), and cancelling/rearranging terms, we get that

$$P_{X|Y}(x|y) = P_{X|Y}(x|y')$$

for all y' for which $P_{XY}(x,y') > 0$. From this, we may conclude that

$$P_X(x) = \sum_{y''\in\mathcal{Y}} P_Y(y'')P_{X|Y}(x|y'') = \sum_{y''\in\mathcal{Y}} P_Y(y'')P_{X|Y}(x|y) = P_{X|Y}(x|y),$$

where the second inequality follows from the fact that $P_{X|Y}(x|y) = P_{X|Y}(x|y'')$ for all x, y, y''. Finally we conclude that $P_{XY}(x,y) = P_{X|Y}(x|y)P_Y(y) = P_X(x)P_Y(y)$. This equality now holds for all $(x,y) \in \mathcal{X} \times \mathcal{Y}$, so X and Y are independent.

The Chain Rule

The chain rule expresses the relation between the conditional entropy and the joint/maringal entropies of the variables involved. We first state and prove the chain rule for two random variables, and then generalize it to n variables.

Proposition: Chain Rule

Let X and Y be random variables. Then

$$H(XY) = H(X) + H(Y|X)\,.$$

Proof hint

We encourage you to try to prove this for yourself. As a starting point, write out the definition of $H(XY)$, and rewrite the terms $P_{XY}\,(x,y)$ into a conditional form in order to relate it to $H(Y|X)$.

Show full proof

The chain rule is a matter of rewriting:

$$\begin{aligned}
H(XY) &= -\sum_{x,y} P_{XY}\,(x,y) \log P_{XY}\,(x,y) \\
&= -\sum_{x,y} P_{XY}\,(x,y) \log\big(P_X(x) P_{Y|X}(y|x)\big) \\
&= -\sum_{x,y} P_{XY}\,(x,y) \log P_X(x) - \sum_{x,y} P_{XY}\,(x,y) \log P_{Y|X}(y|x) \\
&= -\sum_{x} P_X(x) \log P_X(x) - \sum_{x,y} P_{XY}\,(x,y) \log P_{Y|X}(y|x) \\
&= -\sum_{x} P_X(x) \log P_X(x) - \sum_{x} P_X(x) \sum_{y} P_{Y|X}(y|x) \log P_{Y|X}(y|x) \\
&= H(X) + H(Y|X)\,.
\end{aligned}$$

This was to be shown.

The chain rule immediately results in the so-called 'independence bound':

Corollary: Subadditivity (independence bound)

$$H(XY) \leq H(X) + H(Y)\,.$$

Equality holds iff X and Y are independent.

created: 2019-10-21

Proof

$$H(XY) = H(X) + H(Y|X) \leq H(X) + H(Y),$$

where the equality is due to the chain rule and the inequality is due to the upper bound on the conditional entropy that $H(Y|X) \leq H(Y)$ (and equal iff X and Y are independent).

We can naturally generalize the definition of conditional entropy by applying it to the conditional distribution $P_{XY|\mathcal{A}}$; this results in $H(X|Y, \mathcal{A})$, the entropy of X given Y and conditioned on the event $\mathcal{A}$. Since the entropy is a function of the *distribution* of a random variable, the chain rule also holds when conditioning on an event $\mathcal{A}$. Furthermore, it holds that

$$H(X|YZ) = \sum_z P_Z(z) H(X|Y, Z{=}z)\,,$$

which is straightforward to verify. With this observation, we see that the chain rule generalizes as follows.

Corollary: generalized chain rule

Let X, Y and Z be random variables. Then

$$H(XY|Z) = H(X|Z) + H(Y|XZ)\,.$$

Inductively applying the (generalized) chain rule implies that for any sequence $X_1, \ldots, X_n$ of random variables:

$$H(X_1 \cdots X_n) = H(X_1) + H(X_2|X_1) + \cdots + H(X_n|X_{n-1} \cdots X_1)\,.$$

Combining this with the upper bound on the conditional entropy, we see that subadditivity generalizes to

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

Definition: Mutual Information

Definition: Mutual information

Let X and Y be random variables. The mutual information $I(X;Y)$ of X and Y is defined as

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

Thus, in a sense, mutual information reflects the reduction in uncertainty about X when we learn Y. Verify the following properties of the mutual information:

$$\begin{aligned}
I(X;Y) &= H(X) + H(Y) - H(XY) \\
I(X;Y) &= I(Y;X) && (\text{“symmetry”}) \\
I(X;Y) &\geq 0 && (\text{“positivity”}) \\
I(X;Y) &= 0 \text{ iff } X \text{ and } Y \text{ are independent} \\
I(X;X) &= H(X) && (\text{“self-information”})
\end{aligned}$$

Entropy Diagrams for Two Random Variables

The relations between entropy, joint entropy, conditional entropy, mutual information, and conditional mutual information can be summed up visually. Here, we show how to do this when only two random variables are involved: all information-theoretic measures can be nicely represented by means of a Venn-diagram-like **entropy diagram**. From the diagram, one can for instance easily read off the relations $H(X|Y) \leq H(X)$, $I(X;Y) = H(X) + H(Y) - H(XY)$, etc. The case of three random variables will be treated later in this course.

Use the tool below to play around with different distributions of X and Y, so you can see how the distribution affects the different entropic quantities. You can select one of the pre-defined distributions on the top, or use the arrows to move some of the probability weight around. Hover over the formulas on the bottom left to see which parts of the distribution were involved, and which part of the diagram corresponds to that quantity.

Direct link to interactive graph

created: 2019-10-21

Definition: Relative Entropy

We can compare two distributions on the same set $\mathcal{X}$ by considering their relative entropy: this measure reflects how different two distributions are.

Definition: Relative entropy

The relative entropy (or: **Kullback-Leibler divergence**) of two probability distributions P and Q over the same $\mathcal{X}$ is defined by

$$D(P||Q) := \sum_{x \in \mathcal{X} P(x) > 0} P(x) \log \frac{P(x)}{Q(x)},$$

where by convention, $\log \frac{p}{0} = \infty$ for all p.

Note that if $Q(x) = 0$ for some x with $P(x) > 0$, then $D(P||Q) = \infty$.

created: 2019-10-21

Properties of Relative Entropy

Even though relative entropy is always nonnegative (see the theorem below), it is not a proper distance measure, because it is not symmetric and does not satisfy the triangle inequality.

Lemma: Alternative Definition of Mutual Information

The mutual information between X and Y can be expressed in terms of the relative entropy of their distributions as follows:

$$I(X;Y) = D(P_{XY}||P_X \cdot P_Y)$$

Proof

The statement follows by writing out the definitions of mutual information and relative entropy, and rearranging terms.

$$\begin{aligned}
I(X;Y) &= H(X) - H(X|Y) \\
&= -\sum_{x\in\mathcal{X}} P_X(x)\log P_X(x) + \sum_{y\in\mathcal{Y}} P_Y(y) \sum_{x\in\mathcal{X}} P_{X|Y}(x|y)\log P_{X|Y}(x|y) \\
&= -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} P_{XY}(x,y)\log P_X(x) + \sum_{x\in\mathcal{X},y\in\mathcal{Y}} P_{XY}(x,y)\log P_{X|Y}(x|y) \\
&= -\sum_{x\in\mathcal{X},y\in\mathcal{Y}:P_{XY}(x,y)>0} P_{XY}(x,y)\log P_X(x) + \sum_{x\in\mathcal{X},y\in\mathcal{Y}:P_{XY}(x,y)>0} P_{XY}(x,y)\log P_{X|Y}(x|y) \\
&= \sum_{x\in\mathcal{X},y\in\mathcal{Y}:P_{XY}(x,y)>0} P_{XY}(x,y)(-\log P_X(x) + \log P_{X|Y}(x|y)) \\
&= \sum_{x\in\mathcal{X},y\in\mathcal{Y}:P_{XY}(x,y)>0} P_{XY}(x,y)\log \frac{P_{X|Y}(x|y)}{P_X(x)} \\
&= \sum_{x\in\mathcal{X},y\in\mathcal{Y}:P_{XY}(x,y)>0} P_{XY}(x,y)\log \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)} \\
&= D(P_{XY}||P_XP_Y)
\end{aligned}$$

Theorem: Information Inequality

For any two probability distributions P and Q defined on the same $\mathcal{X}$,

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

Equality holds if and only if $P = Q$.

Proof

Left as an exercise. Hint: use Jensen's inequality.

The above lemma and theorem together show that the mutual information is a measure of 'how independent' the variables X and Y are: if $P_{XY} = P_X \cdot P_Y$, the variables are independent and their mutual information is zero.

created: 2019-10-21

Definition: Cross Entropy

Suppose you draw samples from a set $\mathcal{X}$, according do a distribution P, while *thinking* you are drawing those samples according to a distribution Q. How surprised do you expect to be? This surprisal value is expressed by the **cross entropy**, a quantity that is very closely related to the relative entropy.

Definition: Cross Entropy

The cross entropy of two probability distributions P and Q over the same $\mathcal{X}$ is defined by

$$H_C(P,Q) := -\sum_{x\in\mathcal{X}, P(x)>0} P(x)\log Q(x).$$

There are a few things to note about this definition:

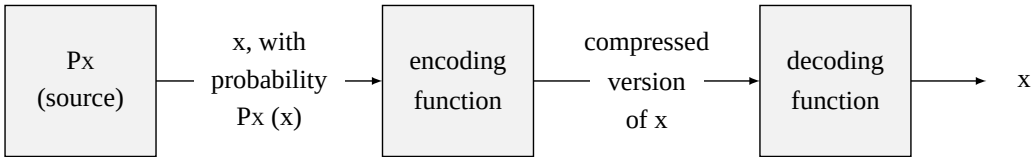
- Note that if $Q(x)=0$ for some x with $P(x)>0$, then $H_C(P,Q)=\infty$.
- Also note that H_C is a function of the *distributions* P and Q. With regular Shannon entropy, we can be sloppy with the notation, and write $H(X)$ instead of the more correct $H(P)$. Here, that sloppy notation would lead to ambiguous expressions.
- In the literature, the notation $H(P,Q)$ is often used to denote the cross entropy. This notation can potentially be confused with the notation for joint entropy, so we use the subscript C to make the distinction explicit.

The cross entropy often pops up in topics related to machine learning: a system usually learns from a set of *training data*, from which it hypothesizes a certain distribution Q (on letter frequencies, cluster sizes, or whatever the system is learning about). When the system is released into the real world, it encounters a (possibly) different distribution P. The cross entropy $H_C(P,Q)$ quantifies how well the system performs in the real world when it was trained on the training set.

created: 2019-10-21

Introduction to Module 03

Suppose we sample x from a distribution P_X with image $\mathcal{X}$. In the context of data compression, P_X is typically called a **source** that emits value $x \in \mathcal{X}$ with probability $P_X(x)$. We want to compress (or encode) symbols x sampled from P_X in such a way that we can later decompress (or decode) it reliably, without losing any information about the value x.



A counting argument shows that it is possible to encode the elements of $\mathcal{X}$ by bit strings of length n, where $n = \lceil \log(|\mathcal{X}|) \rceil$: we simply list all elements of $\mathcal{X}$, and use the (binary) index of x in the list as its encoding. Thus, to store or to transmit an element $x \in \mathcal{X}$, n bits of information always suffice. However, if not all $x \in \mathcal{X}$ are equally likely according to P_X, one should be able to exploit this to achieve codes with shorter *average* length. The idea is to use encodings of varying lengths, assigning shorter codewords to the elements in $\mathcal{X}$ that have higher probabilities, and vice versa.

In the video, Alice and Bob communicate by encoding their messages (dice rolls) from $\mathcal{X} = \{2, 3, \ldots, 12\}$ into a unitary alphabet $\{1\}$, where each 1 stands for a pluck of the wire. For example, the roll 8 is encoded as 111, or three plucks.

Exercise

At the end of the video, Bob gets a better idea. He notices that they can pluck the wire in two different ways that are easy to distinguish: long or short. Can you design a code using this binary alphabet of plucks? How long are your codewords on average?

Show solution

The code on the right is an example of a code that Alice and Bob may use: 0 [illegible] pluck, 1 stands for a long one. Each die roll has a different

codeword, and short codewords are assigned to the most likely outcomes. The expected codeword length is

$$\frac{1}{36}\cdot 3+\frac{2}{36}\cdot 3+\frac{3}{36}\cdot 2+\ldots+\frac{1}{36}\cdot 3=\frac{35}{18}\approx 1.944.$$

So on average, Alice and Bob expect to pluck the wire a little less than two times per die roll they want to communicate. However, if they want to communicate a list of die roll outcomes, they run into a problem: if Alice receives 011, how can she tell whether Bob sent the list [7,4], or [5,6], or even [2]?

Die roll	Codeword
2	011
3	001
4	11
5	01
6	1
7	0
8	00
9	10
10	000
11	010
12	100

This problem is resolved in the code on the right: confusions do not arise even when variable-length lists of messages are sent. The average codeword length is longer, however: roughly 3.306 plucks on average. In this module, you will encounter several algorithms for constructing such codes yourself for any given probability distribution.

Die roll	Codeword
2	11110
3	0010
4	0011
5	100
6	000

created: 2019-10-21

Die roll	Codeword
7	010
8	011
9	101
10	110
11	1110
12	11111

The question we will answer in this module is: how short can codes be in general (on average over repeated samples x from P_X)? We explore both **lossless** codes (where we want to recover the original data with certainty) and **lossy** codes (where with small probability, the data is lost). For now, we will assume that our communication channels are perfect; later in the course, we will introduce channels with some noise.

Definition: Sources and Codes

We start by investigating symbol codes: codes that encode a source P_X, one symbol at a time. Later on, we will also see codes that group the source symbols together into blocks.

Definition: Symbol code

Let P_X be the distribution of a random variable X (with image $\mathcal{X}$), and let $\mathcal{A}$ be a finite set. A symbol code for the source P_X and with alphabet $\mathcal{A}$ is an injective function $C : \mathcal{X} \to \mathcal{A}^*$.

Here, $\mathcal{A}^* = \bigcup_{n \in \mathbb{N}} \mathcal{A}^n \cup \bot$, and $\bot$ is the empty string. That is, $\mathcal{A}^*$ is the set of finite sequences of elements from $\mathcal{A}$: this operation on sets is called the Kleene star.

We often refer to the set of codewords, $\mathcal{C} = \mathrm{im}(C)$, as code and leave the actual encoding function C implicit.

In many instances, the alphabet $\mathcal{A}$ is fixed to be the set $\{0, 1\}$ of size 2. In that case, we speak of a **binary symbol code**. The codewords of a binary code are simply binary strings.

Definition: Codeword length

Let $C : \mathcal{X} \to \mathcal{A}^*$ be an encoding function. For any $x \in \mathcal{X}$, the length $\ell(C(x))$ of the codeword $C(x)$ is the length of the sequence of symbols from $\mathcal{A}$. That is, if $C(x) \in \mathcal{A}^k$, then $\ell(C(x)) = k$.

For practical applications, it is important that the codewords are (on average) short: that way, the transmission or storage of a message is as efficient as possible.

created: 2019-10-21

Definition: Unique decodability

In many situations, we may want to transmit or store more than a single symbol from the source. Instead, we want to transmit or store a list of symbols, or even a potentially infinite stream of symbols.

Definition: Extended symbol code

Let $C : \mathcal{X} \to \mathcal{A}^*$ be a symbol code. The extended code $C^* : \mathcal{X}^* \to \mathcal{A}^*$ is defined by concatenation:

$$C^*(x_1, \ldots, x_n) := C(x_1)| \cdots |C(x_n).$$

The injectivity of C ensures that we can always uniquely decode $C(x)$. However, if one transmits a sequence $x_1, \ldots, x_m \in \mathcal{X}$ (or stores them "sequentially") by sending the concatenation $C(x_1, \ldots, x_n)$, ambiguities may arise, namely in cases where it is possible to parse this long string in two consistent but different ways. Indeed, injectivity of the encoding function per se does not rule out that there exists a positive integer m' and elements $x'_1, \ldots, x'_{m'} \in \mathcal{X}$ such that $C(x_1)| \cdots |C(x_m) = C(x'_1)| \cdots |C(x'_{m'})$. Of course, this problem can be circumvented by introducing a special separation symbol. However, such a symbol might not be available, and maybe even more importantly, even if an additional symbol *is* available, then one can often create a better code by using it as an ordinary alphabet symbol (in addition to 0 and 1) rather than as a special separation symbol. This is why it is interesting to study the following class of symbol codes:

Definition: Uniquely decodable code

A binary symbol code $C : \mathcal{X} \to \{0,1\}^*$ is uniquely decodable if C^* is injective as well.

In the quiz, you will see an example of a code that is not uniquely decodable.

created: 2019-10-21

Definition: Prefix-freeness

One convenient way to guarantee that a code is unique decodable is to require it to be prefix-free:

Definition: Prefix-free code

A binary symbol code $C : \mathcal{X} \to \{0,1\}^*$ is prefix-free (or: **instantaneous**) if for all $x, x' \in \mathcal{X}$ with $x \neq x'$, $C(x)$ is *not* a prefix of $C(x')$.

With a prefix-free encoding, the elements $x_1, \ldots, x_m$ can be uniquely recovered from $C(x_1)|\cdots|C(x_m)$, simply by reading the encoding from left to right one bit at a time: by prefix-freeness it will remain unambiguous as reading continues when the current word terminates and the next begins. This is a loose argument for the following proposition:

Proposition

If a code $\mathcal{C}$ is prefix-free and $\mathcal{C} \neq \bot$ then $\mathcal{C}$ is uniquely decodable.

The other direction does not hold: uniquely decodable codes need not be prefix-free. A prefix-free code is appealing from an efficiency point of view, as it allows to decode "on the fly". For a general uniquely decodable code one may possibly have to inspect all bits in the entire string before being able to even recover the first word.

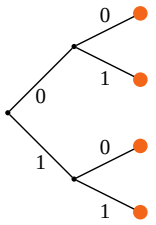
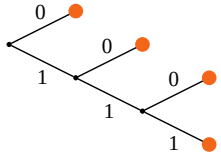
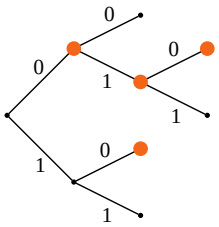
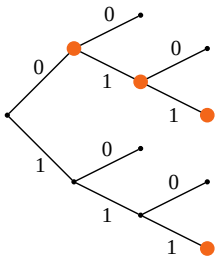
Example

The following are all codes for the source P_X, with $\mathcal{X} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$:

x	$P_X(x)$	$C_1(x)$	$C_2(x)$	$C_3(x)$	$C_4(x)$
a	0.5	00	0	0	0
b	0.25	01	10	010	01
c	0.125	10	110	01	011
d	0.125	11	111	10	111

created: 2019-10-21

These codes can be visualised a as binary trees, with marked codewords, as follows:

			
C_1	C_2	C_3	C_4

Which of these codes are uniquely decodable? Which are prefix-free?

Show solution

C_1 and C_2 are prefix-free, and therefore also uniquely decodable. C_3 is not uniquely decodable, as $C_3(ad) = C_3(b)$. C_4 is not prefix-free, but it is uniquely decodable, since it can be decoded from right to left (it is ``suffix-free''). Note that the binary trees for the prefix-free codes C_1, C_2 only have codewords at the leaves.

created: 2019-10-21

Definition: (Minimal) Average Length

For efficiency reasons, we are often interested in the average (expected) length of a code C:

Definition: Average length

Let $\ell(s)$ denote the length of a string $s \in \{0,1\}^*$. The (average) length of a code C for a source P_X is defined as

$$\ell_C(P_X) := \mathbb{E}[\ell(C(X))] = \sum_{x \in \mathcal{X}} P_X(x)\ell(C(x)) \,.$$

Example

The following are all codes for the source P_X, with $\mathcal{X} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$:

x	$P_X(x)$	$C_1(x)$	$C_2(x)$	$C_3(x)$	$C_4(x)$
a	0.5	00	0	0	0
b	0.25	01	10	010	01
c	0.125	10	110	01	011
d	0.125	11	111	10	111

For the codes above, we obtain the following average codeword lengths: $\ell_{C_1}(P_X) = 2, \ell_{C_2}(P_X) = \ell_{C_4}(P_X) = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = \frac{7}{4} = 1.75$ and $\ell_{C_3}(P_X) = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 3 + \frac{1}{8} \cdot 2 + \frac{1}{8} \cdot 2 = \frac{7}{4} = 1.75$. We see that the codes C_2, C_3, C_4 have a smaller average codeword length, but C_2 and C_4 are preferred over C_3 because their unique decodability. Notice that the individual codeword lengths of codes C_2 and C_4 correspond exactly to the surprisal values of P_X in bits, e.g. $\ell(C_2(b)) = \ell(C_4(b)) = 2 = -\log P_X(b)$. Therefore, the computations of the entropy $H(X)$ and of the average code length $\ell_{C_2}(P_X)$ are exactly the same, and we have that $H(X) = \ell_{C_2}(P_X) = \ell_{C_4}(P_X)$. We will see later that this property characterizes optimal codes.

decide whether this button is necessary, and whether the title of this block should be question instead of example

Definition: Minimal code length

The minimal code length of a source P_X is defined as

$$\ell_{\min}(P_X) := \min_{C \in \mathfrak{C}} \ell_C(P_X)$$

where $\mathfrak{C}$ is some class of codes, for example the set of all prefix-free codes (resulting in $\ell_{\min}^{\text{p.f.}}$), or the set of all uniquely decodable codes (resulting in $\ell_{\min}^{\text{u.d.}}$).

Theorem: Kraft-McMillan Inequality

The two following theorems give necessary and sufficient conditions for the existence of a prefix-free code or a uniquely decodable code for a given set of codeword lengths.

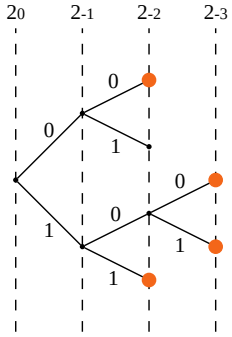
Theorem: Kraft's inequality

Let $\{\ell_1, \ell_2, \ldots, \ell_m\}$ be a set of m positive natural numbers. There exists a prefix-free code with image $\mathcal{C} = \{c_1, \ldots, c_m\}$ and codeword lengths $\ell(c_i) = \ell_i$, if and only if

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1.$$

$\Rightarrow$

For the forward direction, suppose we have a prefix-free code C with image $\mathcal{C} = \{c_1, \ldots, c_m\}$ and codeword lengths $\ell(c_i) = \ell_i$. View this code as a tree, with codewords only on the leaves (but not necessarily all the leaves), and assign a weight of 2^{-d} to every node in the tree at depth d (including the leaves):



Note that the weight of each node is exactly the sum of the weight of its direct children, and thereby that the weight of the root is exactly the weight of all of the leaves. Since every codeword c_i resides on a leaf of depth ℓ_i (but not all leaves are necessarily occupied), the weight of the root is *at least* the sum of all the codeword weights:

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 2^0 = 1.$$

$\Leftarrow$

Shopping at the codeword supermarket as described in the following video: 03MacKayShoppingCodewords.MP4

A stronger version of Kraft's inequality holds as well, this time for uniquely decodable codes:

Theorem: McMillan inequality

For a uniquely decodable code with image $\mathcal{C} = \{c_1, \dots, c_m\}$ and codeword lengths $\ell_i := \ell(c_i)$, it holds that

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1.$$

Proof

Let $\mathcal{C}$ be a uniquely decodable code as in the theorem statement. We can write

$$S := \sum_{c \in \mathcal{C}} \frac{1}{2^{\ell(c)}} = \sum_{\ell=L_{\min}}^{L_{\max}} \frac{n_\ell}{2^\ell}$$

where $L_{\min} = \min_{c \in \mathcal{C}} \ell(c)$, $L_{\max} = \max_{c \in \mathcal{C}} \ell(c)$, and $n_\ell = |\{c \in \mathcal{C} \mid \ell(c) = \ell\}|$. Furthermore, for any $k \in \mathbb{N}$, consider the kth power of S,

$$S^k = \sum_{c_1, \dots, c_k \in \mathcal{C}^k} \frac{1}{2^{\ell(c_1)+\dots+\ell(c_k)}} = \sum_{\ell=kL_{\min}}^{kL_{\max}} \frac{n_\ell^{(k)}}{2^\ell}$$

where $n_\ell^{(k)}$ is defined as
$n_\ell^{(k)} = \left|\left\{(c_1, \dots, c_k) \in \mathcal{C}^k \mid \sum_i \ell(c_i) = \ell(c_1|\dots|c_k) = \ell\right\}\right|$. Note that

$$n_\ell^{(k)} = \sum_{x \in \{0,1\}^\ell} \left|\{(c_1, \dots, c_k) \in \mathcal{C}^k \mid c_1|\dots|c_k = x\}\right| \leq \sum_{x \in \{0,1\}^\ell} 1 = 2^\ell$$

where the inequality follows from the unique decodability of $\mathcal{C}$. Thus, we can conclude that

$$S^k \leq (L_{\max} - L_{\min}) \cdot k$$

for all $k \in \mathbb{N}$, so S^k grows at most linearly in k, from which follows that $S \leq 1$ (for if not, S^k would grow exponentially in k).

Kraft's and McMillan's inequality together lead to the conclusion that the lengths of an optimal prefix-free code and an optimal uniquely decodable code coincide:

Corollary

Let P_X be a source. For every uniquely decodable code C, there exists a prefix-free code C' such that $\ell_C(P_X) = \ell_{C'}(P_X)$. Hence,

$$\ell_{\min}^{\text{p.f.}}(P_X) = \ell_{\min}^{\text{u.d.}}(P_X).$$

From now on, we will just write $\ell_{\min}(P_X)$ to denote either of these measures for average length. A code C for which $\ell_C(P_X) = \ell_{\min}(P_X)$ is called **optimal** for the source P_X.

Visualisation of Symbol Codes as Trees

Play around with the following interactive tool. Thanks a lot toHuub Rutjes for making it!

created: 2019-10-21

Theorem: Shannon's Source-Coding Theorem (Optimal Codes)

We now know that prefix-free codes can achieve the same minimal code lengths for a source P_X as the more general class of uniquely decodable codes. How small is this minimal code length in general? In this section we explore the following relation between the minimal code length and the entropy of the source:

Theorem: Shannon's source-coding theorem (for symbol codes)

For any source P_X, we have the following bounds:

$$H(X) \leq \ell_{\min}(P_X) \leq H(X) + 1.$$

$H(X) \leq \ell_{\min}(P_X)$

The proof relies on Kraft's inequality. Let C be a (uniquely decodable) code, and write ℓ_x for $\ell(C(x))$ as a notational convenience. For the lower bound, we have that

$$\begin{aligned}
H(X) - \ell_C(P_X) &= -\sum_{x\in\mathcal{X}} P_X(x)\log(P_X(x)) - \sum_{x\in X} P_X(x)\ell_x \\
&= \sum_{x\in\mathcal{X}} P_X(x)\left(-\log(P_X(x)) - \log\left(2^{\ell_x}\right)\right) \\
&= \sum_{x\in\mathcal{X}} P_X(x)\log\left(\frac{1}{P_X(x)\cdot 2^{\ell_x}}\right) \\
&\leq \log\left(\sum_{x\in\mathcal{X}} \frac{1}{2^{\ell_x}}\right) && \text{(by Jensen's inequality)} \\
&\leq \log(1) = 0 && \text{(by Kraft's inequality)}
\end{aligned}$$

$\ell_{\min}(P_X) \leq H(X) + 1$

For the upper bound, let us denote by ℓ_x the surprisal value in bits rounded up to the next integer, i.e. for any $x \in \mathcal{X}$,

$$\ell_x := \left\lceil \log \frac{1}{P_X(x)} \right\rceil,$$

and note that

$$\sum_{x\in\mathcal{X}} 2^{-\ell_x} \leq \sum_{x\in\mathcal{X}} 2^{-\log\frac{1}{P_X(x)}} = \sum_{x\in\mathcal{X}} P_X(x) = 1.$$

Therefore, by Kraft's inequality, there exists a prefix-free code C such that $\ell(C(x)) = \ell_x$ for all $x \in \mathcal{X}$. This code satisfies

$$\begin{aligned}\ell_C(P_X) &= \sum_{x\in\mathcal{X}} P_X(x)\ell_x \\ &\leq \sum_{x\in\mathcal{X}} P_X(x)\left(\log\frac{1}{P_X(x)}+1\right) \\ &= -\sum_{x\in\mathcal{X}} P_X(x)\log P_X(x) + \sum_{x\in\mathcal{X}} P_X(x) \\ &= H(X)+1.\end{aligned}$$

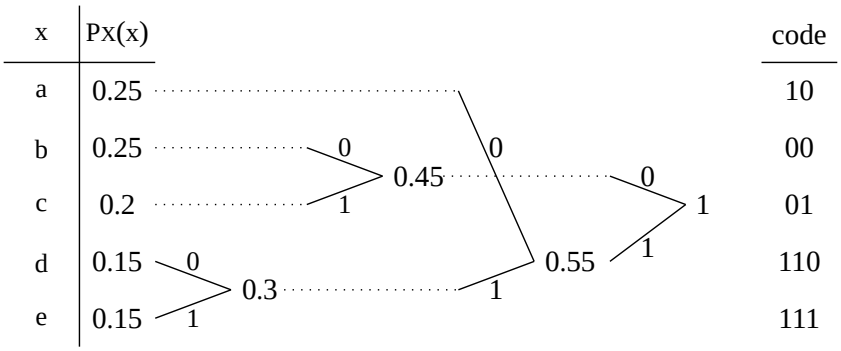
We have thus constructed a code C with $\ell_C(P_X) \leq H(X)+1$, so $\ell_{\min}(P_X) \leq H(X)+1$.

Definition: Huffman Codes

Shannon's source-coding theorem shows us that in theory, the minimal code length for a source P_X is roughly $H(X)$. In this section we will investigate **Huffman codes**, which provide an explicit and neat construction for optimal prefix-free codes. A binary Huffman code for a source P_X is constructed by iteratively pairing the two symbols with the smallest probability together, building a binary tree on the way. This is best explained by example:

Example: Binary Huffman code

Let the random variable X be given with $\mathcal{X} = \{a, b, c, d, e\}$ and $P_X(a) = P_X(b) = 0.25$, $P_X(c) = 0.2$, and $P_X(d) = P_X(e) = 0.15$. The following is a binary Huffman code for P_X:



We build up the tree from left to right, pairing the symbols (or groups of symbols) with smallest (combined) probabilities at every step. The codeword for every symbol is then determined by following the branches of the tree *from right to left* until the symbol is reached. Note that this way, the symbols with the smallest probabilities get assigned the longest codewords (paths). The average codeword length for this code is

$$0.25 \cdot 2 + 0.25 \cdot 2 + 0.2 \cdot 2 + 0.15 \cdot 3 + 0.15 \cdot 3 = 2.3.$$

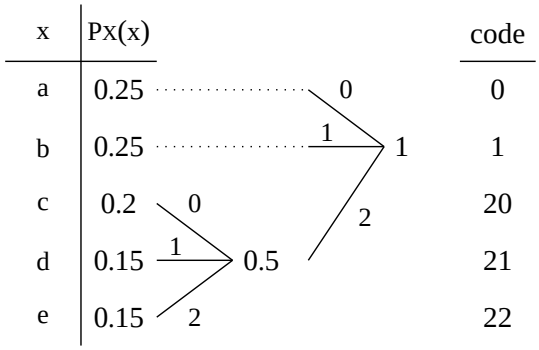
This very close to the entropy $H(X) \approx 2.285$. The average codeword length lies between $H(X)$ and $H(X) + 1$.

Definition: Ternary Huffman Codes

We saw how to construct *binary* Huffman codes. We can also generate Huffman codes for larger alphabets, resulting in ternary, quaternary, or, more generally, d-**ary Huffman codes**.

Example: Ternary Huffman code

We build a Huffman code with the alphabet $\{0,1,2\}$ for the same distribution as in the previous example.



Now, use the above procedure to construct a ternary code for the source P_X with $\mathcal{X} = \{a,b,c,d,e,f\}$ and $P_X(a) = P_X(b) = 0.25$, $P_X(c) = 0.2$, $P_X(d) = P_X(e) = P_X(f) = 0.1$. Can you find another code with a smaller average codeword length?

We have to be careful, because with an alphabet size of greater than 2, the above procedure does not always give an optimal code! In fact, a d-ary code is only optimal if $|\mathcal{X}|$ is of the form $k(d-1)+1$ for some $k \in \mathbb{N}$. This ensures that at every step, we can combine exactly d symbols to use the alphabet at full capacity. The ternary code in the example above is optimal because $|\mathcal{X}| = 5 = 2(3-1)+1$, but the code you constructed in the exercise is not. To remedy this, one can add one or more `dummy' symbols to the source (each with probability zero) until an appropriate size of $\mathcal{X}$ of the form $|\mathcal{X}| = k(d-1)+1$ for some $k \in \mathbb{N}$ is reached. The codewords for those dummy symbols are discarded at the end. It turns out that Huffman codes indeed have optimal code length (see Cover/Thomas, Section 5.8)

Ternary code applet

created: 2019-10-21

Advantages and Disadvantages of Huffman Codes

It turns out that Huffman codes indeed have optimal code length.

Theorem: Optimality of Huffman codes

Let P_X be a source, and let C^* be the associated Huffman code. For any other uniquely decodable code C' with the same alphabet,

$$\ell_{C^*}(P_X) \leq \ell_{C'}(P_X).$$

Proof

see Cover/Thomas, Section 5.8

As we have seen, the average codeword length of Huffman codes is theoretically optimal. However, Huffman codes (and symbol codes in general) still have a number of disadvantages:

- When compressing, for example, an English text symbol-by-symbol, the probability distribution for each position may depend on the string of text that precedes it: for example, the letter ***n*** is a lot more likely than the letter ***a*** if it comes after the string ***informatio***. Given this change of distribution, the Huffman code may not produce the shortest possible code. This can be resolved by recomputing the Huffman code after every symbol, but this results in a lot of overhead.
- The average codeword length is upper bounded by $H(X) + 1$. This additive cost of 1 bit is fine when $H(X)$ is very large, but can be a significant overhead when $H(X)$ is small itself.

created: 2019-10-21

Hangman

Play the following game, try to complete at least 5 sentences!

If you are tired of it, you can look at the stats, but it's going to spoil all the fun for playing. Clearly, the number of tries for letters depends heavily on the context! Therefore, the distribution of letters changes a lot with context, so Huffman codes are not well-suited for this purpose.

Thanks a lot toHuub Rutjes for making the game!

Definition: Binary Representation & Binary Intervals

Definition: Standard binary representation

The standard binary representation of a real number $r \in [0,1)$ is a (possibly infinite) string of bits $c_1 c_2 \cdots$ such that

$$r = \sum_i c_i \cdot 2^{-i},$$

where by convention, 0 is represented by the string 0.

Not all reals in $[0,1)$ have a finite representation, but any interval $[a,b)$ with $0 \leq a < b \leq 1$ contains at least one number with a finite binary representation.

Example

The following table lists some numbers $r \in [0,1)$ and their standard binary representation.

r	binary representation of r
$1/2$	1
$1/3$	$01010101\cdots$
$1/4$	01
$3/4$	11
$13/16$	1101
$13/32$	01101

Note that 1101 is also the binary form of the natural number 13. Adding a 0 on the left divides the represented value by 2.

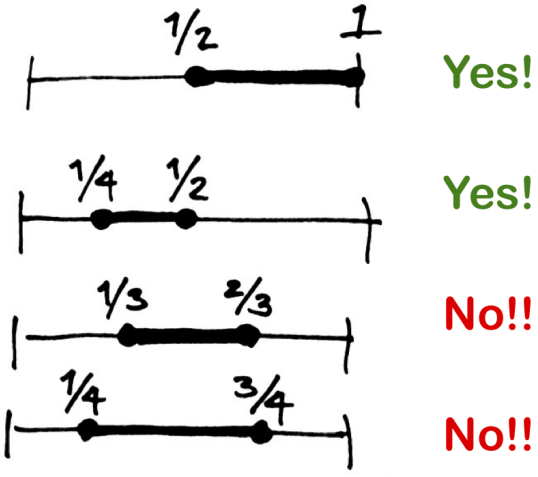
Definition: Binary interval

A binary interval is an interval of the form

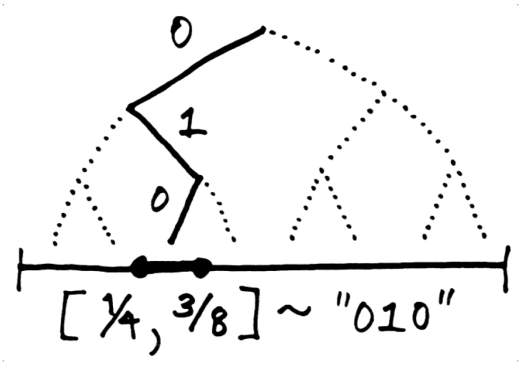
$$\left[\frac{s}{2^\ell}, \frac{s+1}{2^\ell}\right)$$

with $s, \ell \in \mathbb{N}$ and $0 \leq s < 2^\ell$.

created: 2019-10-21



The **name** of the interval is the binary representation of s (as a natural number) padded with zeroes on the left to reach length ℓ. The name can also be interpreted as the path to follow from the root in order to reach the interval as follows:



[Images by Mathias Madsen, thanks a lot!]

Arithmetic Codes

Visualization using "language model" of controller presses (X,Y,A,B)

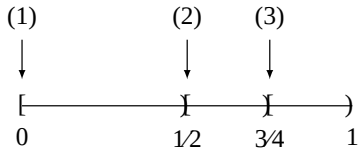
In this section, we study a different kind of code that can handle context-dependent distributions, unlike the Huffman code. Binary representations of numbers in the interval $[0,1)$ as studied in the previous exercise give rise to a very elegant code:

Definition: Arithmetic code

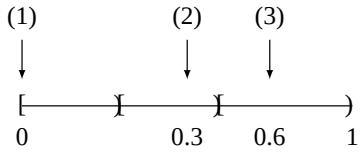
Given a source P_X with $\mathcal{X} = \{x_1, .., x_m\}$, construct the arithmetic code as follows. Divide the interval $[0,1)$ into disjoint subintervals $I_{x_j} = [a_j, a_{j+1})$, where $a_1, \ldots, a_{m+1}$ are defined such that $a_{j+1} - a_j = P_X(x_j)$, and $a_1 = 0, a_{m+1} = 1$. The encoding $AC(x_j)$ of the element x_j is the (shortest possible) standard binary representation of some number in the interval I_{x_j}.

Example

Let X be a random variable with $\mathcal{X} = \{1,2,3\}$ and $P_X(1) = \frac{1}{2}$, $P_X(2) = P_X(3) = \frac{1}{4}$. The arithmetic code is constructed by first determining the intervals:



This image results in the arithmetic code $\mathcal{C}$ with $\mathcal{C}(1) = 0$ (the representation of 0), $\mathcal{C}(2) = 1$ (the representation of $\frac{1}{2}$), $\mathcal{C}(2) = 11$ (the representation of $\frac{3}{4}$). For P_X, the codewords happen to fall exactly *on* the boundaries of the intervals. This is not always the case, however. The same code would have resulted from this procedure if we started with the random variable Y with $\mathcal{Y} = \{1,2,3\}$ and $P_Y(1) = P_Y(2) = 0.3$ and $P_Y(3) = 0.4$:



Proposition

For any (X, P_X), the arithmetic code has average length $\ell_{AC}(P_X) \leq H(X) + 1$.

Proof

Let $x \in \mathcal{X}$, and define $\ell_x := \lceil \log(1/P_X(x)) \rceil$ to be the rounded surprisal value of x. Then

$$2^{-\ell_x} = 2^{-\lceil \log(1/P_X(x)) \rceil} \leq 2^{-\log(1/P_X(x))} = 2^{\log P_X(x)} = P_X(x).$$

Therefore, since the size of the interval I_x is $P_X(x)$, there must exist an integer $0 \leq s_x < 2^{\ell_x}$ such that $s_x \cdot 2^{-\ell_x}$ lies in the interval I_x. This number $s_x \cdot 2^{-\ell_x}$ has a binary representation of length $\ell_x \leq -\log P_X(x) + 1$. Repeating this argument for every x, we obtain

$$\ell_{AC}(P_X) = \mathbb{E}[\ell(AC(X))] = \sum_x P_X(x)\ell(AC(x)) \leq \sum_x P_X(x)(-\log P_X(x) + 1) = H(X) + 1.$$

As we can see from the example above, this construction for arithmetic codes does not necessarily yield prefix-free codes. However, at the expense of one extra bit of code (on average), the construction can be adapted into a prefix-free code. One example of this is the **Shannon-Fano-Elias code** (see Cover/Thomas, Section 5.9 or Wikipedia), which provides a more sophisticated way of selecting a number within each interval than simply selecting the number with the shortest binary representation. This alternative selection procedure ensures prefix-freeness. Another option is to select *binary intervals* within each interval:

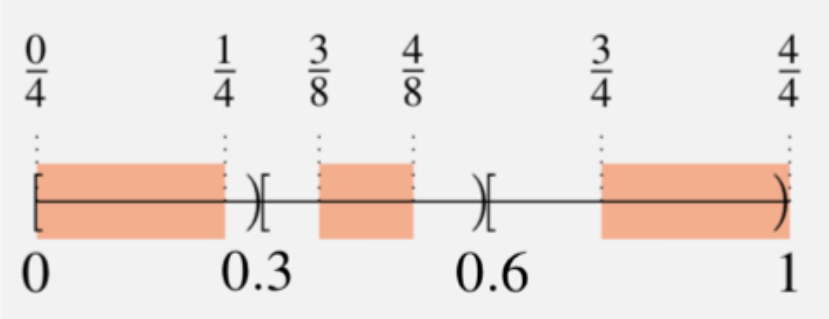
Definition: Arithmetic code (prefix-free version)

A prefix-free arithmetic code is identical to the definition above, except that the encoding $AC^{pf}(x_j)$ of the element x_j is now the name of a largest binary interval that fits entirely in I_{x_j}.

Similarly to the proposition above, it can be shown that for any source P_X, $\ell_{AC^{pf}}(P_X) \leq H(X) + 2$. Note that we get prefix-freeness only at the expense of an extra bit on average.

Example

Let Y be the random variable as in the example above, that is, $P_Y(1) = P_Y(2) = 0.3$ and $P_Y(3) = 0.4$. The prefix-free code for Y is constructed as follows:

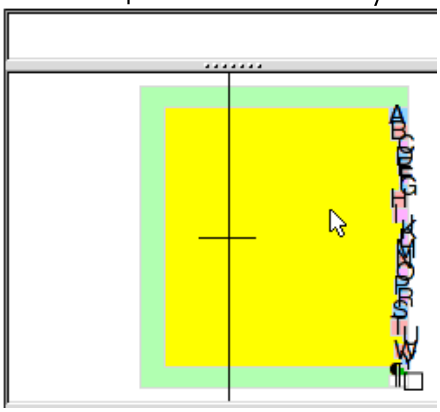


This results in the codewords $\mathcal{C}(1) = 00$, $\mathcal{C}(2) = 011$, and $\mathcal{C}(3) = 11$.

The arithmetic code is slightly less efficient than the Huffman code in terms of average codeword length. A big advantage is the way it is able to adapt to changing distributions, such as when we are encoding a stream of English text. Suppose we are given the (not necessarily i.i.d.) random variables $X_1, X_2, \ldots, X_n$, and we want to encode the source $P_{X_1 X_2 \cdots X_n}$. We start by dividing the interval $[0, 1)$ into subintervals according to P_{X_1}. If, for example, the event $X_1 = b$ happens, we zoom into the interval corresponding to b, and subdivide *that* interval according to $P_{X_2|X_1}$, so that the sizes of these intervals add up to $P_{X_1}(b)$.

The concept of arithmetic coding is exploited as an accessibility tool in the keyboard alternative Dasher, invented by the group of David MacKay at Cambridge University, UK. Try to download and play with it, it's great fun!

Arithmetic Codes | Information Theory



created: 2019-10-21

Introduction to Module 04

This module consists of two separate topics.

The first topic builds on the previous module, where we looked at coding a source into short codewords. Instead of encoding each symbol that comes from the source separately, we will now code entire blocks at the same time: we wait until the source has emitted, say, n symbols, and then assign that block of n symbols a single codeword. Can we make shorter codes in this way?

It turns out the answer is yes: in some cases, the expected codeword length can shrink slightly. A code that achieves this will work as follows. Divide all possible source strings of length n into two groups: "typical" and "atypical". A typical source string will be assigned a short codeword, whereas an atypical source string will be assigned a longer codeword. Ideally, the source will almost always emit typical strings, but the set of typical strings (the "typical set") will be very small, so that you can assign very short codewords to its elements. This balance will be formalized in this module when we define typical sets. The concept of typicality will play an important role again in the final module of this course, when we prove Shannon's noisy-channel coding theorem.

The second topic is about *hiding* information: we will see some simple examples of encryption schemes, one of which is perfectly secure (we will define what that means later). We will consider the following question: how many bits of key are required to perfectly hide all the information in a message? Perhaps unsurprisingly, the answer will be related to the entropy of the message.

To understand perfectly secure encryption, we will complete our picture of entropy-related quantities by introducing conditional mutual information.

created: 2019-10-21

Convergence of Random Variables

The following definition of converging random variables may remind you of a converging sequence of numbers. Recall that a sequence $x_1, x_2, x_3, \ldots$ of numbers converges to x if $\forall \epsilon > 0 \; \exists n_0 \; \forall (n \geq n_0) : |x_n - x| < \epsilon$. We denote this by writing $x_n \xrightarrow{n \to \infty} x$.

Definition: Converging random variables

A sequence $X_1, X_2, X_3, \ldots$ of real random variables converges to a random variable X, if it satisfies one of the following definitions:

in probability	(notation $X_n \xrightarrow{p} X$)	if $\forall \epsilon > 0$, $P[\lvert X_n - X \rvert > \epsilon] \xrightarrow{n \to \infty} 0$
		"As n increases, the distribution of X_n gets closer and closer to that of X."
in mean square	(notation $X_n \xrightarrow{m.s.} X$)	if $\mathbb{E}[(X_n - X)^2] \xrightarrow{n \to \infty} 0$
		"As n increases, the expected (square of the) difference between X_n and X diminishes."
almost surely	(notation $X_n \xrightarrow{a.s.} X$)	if $P[\lim_{n \to \infty} X_n = X] = 1$
		"Any event ω for which X_n does *not* approach the distribution X has zero probability."

The definition of $X_n \xrightarrow{a.s.} X$ can be interpreted as $P[\{\omega \in \Omega \mid X_n(\omega) \xrightarrow{n \to \infty} X(\omega)\}] = 1$.

In general, the following implications hold (although their converses do not):

$$\begin{aligned} X_n \xrightarrow{m.s.} X \quad &\Rightarrow \quad X_n \xrightarrow{p} X \\ X_n \xrightarrow{a.s.} X \quad &\Rightarrow \quad X_n \xrightarrow{p} X \end{aligned}$$

created: 2019-10-21

The Weak Law of Large Numbers

Recall the weak law of large numbers from the first homework set. We rephrase it here in terms of converging random variables. It states that if we sample several times from the same distribution, the average converges (in probability) to the expected value of the distribution.

Theorem: Weak Law of Large Numbers

Let $X_1, X_2, \ldots$ be real i.i.d. random variables with mean $\mu = \mathbb{E}[X_i]$ and variance $\sigma^2 = \mathbb{E}[(X_i - \mu)^2] < \infty$. Define the random variables

$$S_n := \frac{1}{n} \sum_{i=1}^{n} X_i.$$

Then $S_n \xrightarrow{p} \mu$, where we interpret μ as the constant random variable that is μ with probability 1.

created: 2019-10-21

The Asymptotic Equipartition Property (AEP)

The weak law of large numbers has an entropy variant, which follows almost directly:

Theorem: Asymptotic Equipartition Property (AEP)

Let $X_1, X_2, X_3, \ldots$ be i.i.d. random variables with distribution P_X. Then

$$-\frac{1}{n}\log P_{X_1 \cdots X_n}(X_1, \ldots, X_n) \xrightarrow{p} H(X).$$

(Note that $P_{X_1 \cdots X_n}(X_1, \ldots, X_n)$ is itself a random variable, and $H(X)$ can be regarded as a constant random variable.)

Proof

Since the variables X_i are independent, so are the random variables $\log P_X(X_i)$. Then

$$\begin{aligned} -\frac{1}{n}\log P_{X_1 \cdots X_n}(X_1, \ldots, X_n) &= -\frac{1}{n}\sum_{i=1}^{n} \log P_X(X_i) \\ &\xrightarrow{p} -\mathbb{E}[\log P_X(X_i)] = H(X) \end{aligned}$$

by the weak law of large numbers.

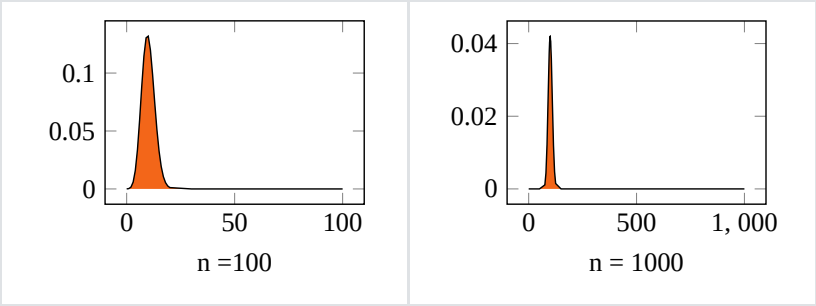
In terms of surprisal values, the AEP states that your surprisal value, averaged over all samples, will eventually approach the entropy $H(X)$ of a single sample.

created: 2019-10-21

Definition: Typical Set

Example: Biased coin flip

Consider flipping a biased coin, with probability of heads being $P_X(\mathtt{h}) = 0.1$ and probability of tails being $P_X(\mathtt{t}) = 0.9$, and counting the number of heads that come up. The random variable Y describing this number is distributed according to the binomial$(n, P_X(\mathtt{h}))$ distribution, where n is the number of coin flips. Below, the distribution of Y is plotted for $n = 100$ and $n = 1000$:



We see that the variance of the sample mean, $\mathrm{Var}[\frac{1}{n}Y] = \frac{0.1(1-0.1)}{n}$, decreases as the number of samples increases. The weight of the distribution becomes centered around an increasingly narrow set of outcomes. Sequences of coin flips that result in an unusual number of heads become increasingly rare: almost all sequences are "typical".

The above example leads us to defining the following subset of the image of a random variable:

Definition: Typical set

The typical set $A_\epsilon^{(n)}$ with respect to P_X is the set of strings $(x_1, \ldots, x_n) \in \mathcal{X}^n$ such that

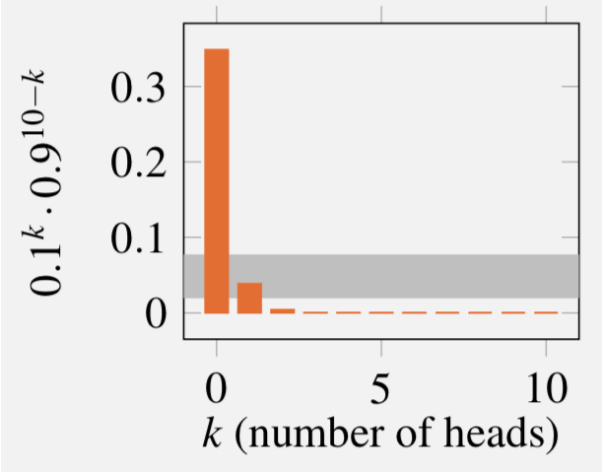
$$2^{-n(H(X)+\epsilon)} \leq P_{X^n}(x_1, \ldots, x_n) \leq 2^{-n(H(X)-\epsilon)},$$

where $P_{X^n}(x_1, \ldots, x_n) = \prod_{i=1}^n P_X(x_i)$.

The typical set is relatively small, but contains almost all of the probability mass.

Example

Consider again the experiment of flipping n biased coins, where the probability of observing "heads" is p, giving $H(X) = h(p)$. The graph below lets you investigate the typical sets for different values of n, p, and ϵ. [Unfortunately, the tool is not quite ready yet, we will supply it as soon as it works...]



On the horizontal axis, the different possible numbers of heads (0 to n) for a sequence x are shown; this is the 'weight' of the sequence x. The vertical axis shows the probability of observing that specific sequence x. (Note that that is *not* the probability of the event "observe $\mathrm{weight}(x)$ heads", because such an event contains many different sequences). The gray area represents the range of probabilities that cause a sequence to fall into the typical set.

For example, set $p = 0.1$, $n = 10$ and $\epsilon = 0.1$. The typical set contains those elements that have probability between $2^{-10(h(0.1)+0.1)} \approx 0.0194$ and $2^{-10(h(0.1)-0.1)} \approx 0.0775$. Using the fact that a sequence with k heads occurs with probability $p^k \cdot (1-p)^{10-k}$, we can compute that the typical set consists of all sequences with exactly 1 heads and 9 tails (those sequences all have probability approximately 0.0387). We see that indeed, the bar with $\mathrm{weight}(x) = 1$ falls into the gray area. Note that in particular, the all-tails outcome ($\mathrm{weight}(x) = 0$) is *not* included in the typical set, even though is by far the most likely outcome. This is because there is only one sequence with $\mathrm{weight}(x) = 0$, and already 10 sequences with $\mathrm{weight}(x) = 1$: together, those 10 sequences have a higher probability mass (0.387) than the single all-tails outcome (0.349). Hence, the typical set still contains a lot of the probability mass.

This effect intensifies as we increase n: for $n = 1000$, the probability of the typical set (containing elements with 69--131 heads) is $P[A_\epsilon^{(1000)}] \approx 0.999$. The `typical' sequences, where roughly one in ten tosses results in a heads, make up almost all of the probability. The high-probability sequences, those with fewer heads, are so scarce that their total probability mass is tiny.

created: 2019-10-21

Properties of Typical Sets

In the coin-flipping example on the previous page, the typical set eventually contained almost all of the probability. The following proposition states that this is a general property of typical sets. The proposition also bounds the size of the typical set.

Proposition

A typical set $A_\epsilon^{(n)}$ satisfies the following:

1. For all $(x_1, \ldots, x_n) \in A_\epsilon^{(n)}$,
$$H(X) - \epsilon \leq -\frac{1}{n} \log P_{X^n}(x_1, \ldots, x_n) \leq H(X) + \epsilon.$$
2. $P[A_\epsilon^{(n)}] > 1 - \epsilon$ (for large enough n).
3. $|A_\epsilon^{(n)}| \leq 2^{n(H(X)+\epsilon)}$.
4. $|A_\epsilon^{(n)}| \geq (1-\epsilon) 2^{n(H(X)-\epsilon)}$ (for large enough n).

Proof

1. This is immediate from the definition (take the logarithm and divide by $-n$, thereby reversing the inequalities).
2. This follows from the Asymptotic Equipartition Property: for all $\epsilon > 0$,
$$P[|-\frac{1}{n} \log P_{X^n}(X_1, \ldots, X_n) - H(X)| > \epsilon] \xrightarrow{n \to \infty} 0,$$
that is,
$$\forall (\epsilon > 0)\ \forall (\delta > 0)\ \exists n_0\ \forall (n \geq n_0)\ P[|-\frac{1}{n} \log P_{X^n}(X_1, \ldots, X_n) - H(X)| \leq \epsilon] > 1 - \delta.$$
By choosing $\delta := \epsilon$, the result follows from the first property.
3. First, observe that
$$1 = \sum_{\vec{x} \in \mathcal{X}^n} P_{X^n}(\vec{x}) \geq \sum_{\vec{x} \in A_\epsilon^{(n)}} P_{X^n}(\vec{x}) \geq |A_\epsilon^{(n)}| \cdot 2^{-n(H(X)+\epsilon}},$$
where the last inequality follows by the definition of typicality. The claim follows by multiplying both sides of the equation by $2^{n(H(X)+\epsilon)}$.
4. By Property 2, we can choose an n large enough so that
$$1 - \epsilon < P[A_\epsilon^{(n)}] = \sum_{\vec{x} \in A_\epsilon^{(n)}} P_{X^n}(\vec{x}) \leq |A_\epsilon^{(n)}| \cdot 2^{-n(H(X)-\epsilon)},$$

where again, the last inequality follows by the definition of typicality.

Source Coding using Typical Sets

The concept of typical sets is useful for designing codes for a source P_X. Instead of encoding the source symbol-by-symbol, we will encode the source symbols in blocks of n symbols at a time.

This strategy can be used to design either a lossy or a lossless code. For a lossy code, we notice that with overwhelming probability, a sequence of n iid samples from P_X is typical, so it suffices to assign binary labels of length (at most) $\lceil n(H(X)+\epsilon)\rceil$ to the elements of $A_\epsilon^{(n)}$, and assign some constant (dummy) codeword to all elements outside of the set. Decoding this dummy codeword will result in an error (data loss), but this error occurs with probability at most ϵ.

The above scheme can be extended to a lossless version by assigning longer labels to the elements outside of $A_\epsilon^{(n)}$, for example binary labels of length $\lceil \log|\mathcal{X}|^n\rceil = \lceil n\log|\mathcal{X}|\rceil$. An extra 'flag' bit is needed to indicate whether the element is inside or outside the typical set. For large enough n, this code is quite efficient:

Theorem

Let $X_1,\ldots,X_n$ be i.i.d. real random variables with respect to the set $\mathcal{X}$, and distributed according to P_X. Let $\epsilon>0$. Then there exists a lossless code $\mathcal{X}^n \to \{0,1\}^*$ such that, for sufficiently large n, $\mathbb{E}[\frac{1}{n}\ell(X^n)] \le H(X)+\epsilon$.

Proof

Consider the code described above: the code consist of a flag bit (indicating whether or not the element is inside the typical set), followed by either a short label (for elements in the typical set) or a longer one (for elements outside of it). Let $\epsilon'>0$ (we will specify the value of ϵ' later). Let n be large enough such that $P[A_{\epsilon'}^{(n)}] > 1-\epsilon'$ (see the second property of typical sets). Then

$$\begin{aligned}\mathbb{E}[\ell(X^n)] &= \sum_{\vec{x}\in\mathcal{X}^n} P_{X^n}(\vec{x})\ell(\vec{x}) \\ &= \sum_{\vec{x}\in A_{\epsilon'}^{(n)}} P_{X^n}(\vec{x})\ell(\vec{x}) + \sum_{\vec{x}\notin A_{\epsilon'}^{(n)}} P_{X^n}(\vec{x})\ell(\vec{x}) \\ &\leq P[A_{\epsilon'}^{(n)}]\cdot(\lceil n(H(X)+\epsilon')\rceil+1) + P[\overline{A_{\epsilon'}^{(n)}}]\cdot(\lceil n\log|\mathcal{X}|\rceil+1) \\ &\leq P[A_{\epsilon'}^{(n)}]\cdot(n(H(X)+\epsilon')+2) + P[\overline{A_{\epsilon'}^{(n)}}]\cdot(n\log|\mathcal{X}|+2) \\ &\leq n(H(X)+\epsilon') + \epsilon'\cdot n\log|\mathcal{X}| + 2 \\ &= n(H(X)+\epsilon),\end{aligned}$$

where $\epsilon = \epsilon' + \epsilon'\log|\mathcal{X}| + \frac{2}{n}$ (note that ϵ can be made arbitrarily small by choosing ϵ' and n wisely). The +1 in the first inequality is a consequence of the `flag' bit.

For large enough blocks of symbols, the flag bit becomes irrelevant. Typical sets thus allow the construction of an efficient code without the 1 bit of overhead that symbol codes may necessarily have. However, this efficiency is only guaranteed for 'sufficiently large n', a rather theoretical condition that may not be achievable in practice.

We conclude this chapter by showing that the typical set is in a sense 'optimal', i.e. that picking a smaller set instead of the typical set does not allow for much shorter codewords on average in a lossy setting, not even if we allow rather large error probabilities by allowing about half of the elements to lie outside of the typical set.

Let $B_\delta^{(n)}$ denote the smallest subset of $\mathcal{X}^n$ such that $P[B_\delta^{(n)}] > 1-\delta$ (for some parameter $\delta > 0$). $B_\delta^{(n)}$ can be explicitly constructed by, for example, ordering $\mathcal{X}^n$ in order of decreasing probability, and adding elements to $B_\delta^{(n)}$ until the probability threshold of $1-\delta$ is reached. The following theorem states that even for large values of δ, we still need almost $nH(X)$ bits to denote an element from $B_\delta^{(n)}$.

Theorem

Let $X_1, \ldots, X_n$ be i.i.d. random variables distributed according to P_X. For any $\delta < \frac{1}{2}$, and any $\delta' > 0$, if $P[B_\delta^{(n)}] > 1-\delta$, then

$$\frac{1}{n}\log|B_\delta^{(n)}| > H(X) - \delta',$$

for sufficiently large n.

Proof

Let $\delta, \epsilon < \frac{1}{2}$, and consider some $B_\delta^{(n)}$ such that $P[B_\delta^{(n)}] > 1-\delta$. By the second property of typical sets, $P[A_\epsilon^{(n)}] > 1-\epsilon$, for large enough n. Thus, by the union bound,

$$\begin{aligned}
1-\epsilon-\delta &< 1 - P[\overline{A_\epsilon^{(n)}}] - P[\overline{B_\delta^{(n)}}] \\
&\leq 1 - P[\overline{A_\epsilon^{(n)}} \cup \overline{B_\delta^{(n)}}] \\
&= P[A_\epsilon^{(n)} \cap B_\delta^{(n)}] \\
&= \sum_{\vec{x} \in A_\epsilon^{(n)} \cap B_\delta^{(n)}} P_{X^n}(\vec{x}) \\
&\leq \sum_{\vec{x} \in A_\epsilon^{(n)} \cap B_\delta^{(n)}} 2^{-n(H(X)-\epsilon)} \\
&= |A_\epsilon^{(n)} \cap B_\delta^{(n)}| \cdot 2^{-n(H(X)-\epsilon)} \\
&\leq |B_\delta^{(n)}| \cdot 2^{-n(H(X)-\epsilon)}.
\end{aligned}$$

Rearranging this expression and taking the logarithm, we get

$$H(X) - \epsilon + \frac{1}{n}\log(1-\epsilon-\delta) < \frac{1}{n}\log|B_\delta^{(n)}|.$$

If we now set $\delta' := \epsilon - \frac{1}{n}\log(1-\epsilon-\delta)$, then

$$H(X) - \delta' < \frac{1}{n}\log|B_\delta^{(n)}|,$$

as desired. Observe that we can make the expression for δ' as small as desired by choosing a small enough $\epsilon > 0$ and a large enough n, even if δ is rather large.

Definition: Conditional Mutual Information

Applying the definition of mutual information to the conditional distribution $P_{XY|\mathcal{A}}$ naturally defines $I(X;Y|\mathcal{A})$, the mutual information of X and Y conditioned on the event $\mathcal{A}$:

Definition: Conditional mutual information

Let X, Y, Z be random variables. Then the conditional mutual information of X and Y given Z is defined as

$$I(X;Y|Z) = \sum_z P_Z(z) I(X;Y|Z{=}z)\,,$$

with the convention that the corresponding argument in the summation is 0 for z with $P_Z(z) = 0$.

Conditional mutual information has properties similar to the ones we saw for mutual information:

$$\begin{aligned} I(X;Y|Z) &= I(Y;X|Z) \\ I(X;Y|Z) &\geq 0 \\ I(X;Y|Z) &= 0 \text{ iff } X \text{ and } Y \text{ are independent given } Z. \end{aligned}$$

Furthermore, the previous bounds $H(X) \geq 0$, $H(X|Y) \geq 0$, and $I(X;Y) \geq 0$, can all be seen as special cases of $I(X;Y|Z) \geq 0$. These bounds, and any bound they imply, are called **Shannon inequalities**. It is important to realize that $I(X;Y|Z)$ may be larger or smaller than (or equal to) $I(X;Y)$.

The following is sometimes used as definition of $I(X;Y|Z)$: verify it for yourself using the definition above.

Alternative definition

Let X, Y, Z be random variables. Then

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

Proof

$$\begin{aligned} I(X;Y|Z) &= \sum_{z\in\mathcal{Z}} P_Z(z) I(X;Y|Z=z) && \text{(by definition)} \\ &= \sum_{z\in\mathcal{Z}} P_Z(z)(H(X|Z=z) - H(X|Y,Z=z)) && \text{(definition of mutual information)} \\ &= \sum_{z\in\mathcal{Z}} P_Z(z) H(X|Z=z) - \sum_{z\in\mathcal{Z}} P_Z(z) H(X|Y,Z=z) \\ &= H(X|Z) - H(X|YZ). \end{aligned}$$

The Chain Rule for Mutual Information

Similarly to the chain rule for entropy, we can prove a chain rule for mutual information:

Corollary: Chain rule for mutual information

Let W, X, Y and Z be random variables. Then

$$I(WX;Y|Z) = I(X;Y|Z) + I(W;Y|ZX)\,.$$

Proof hint

Apply the generalized chain rule.

Show full proof

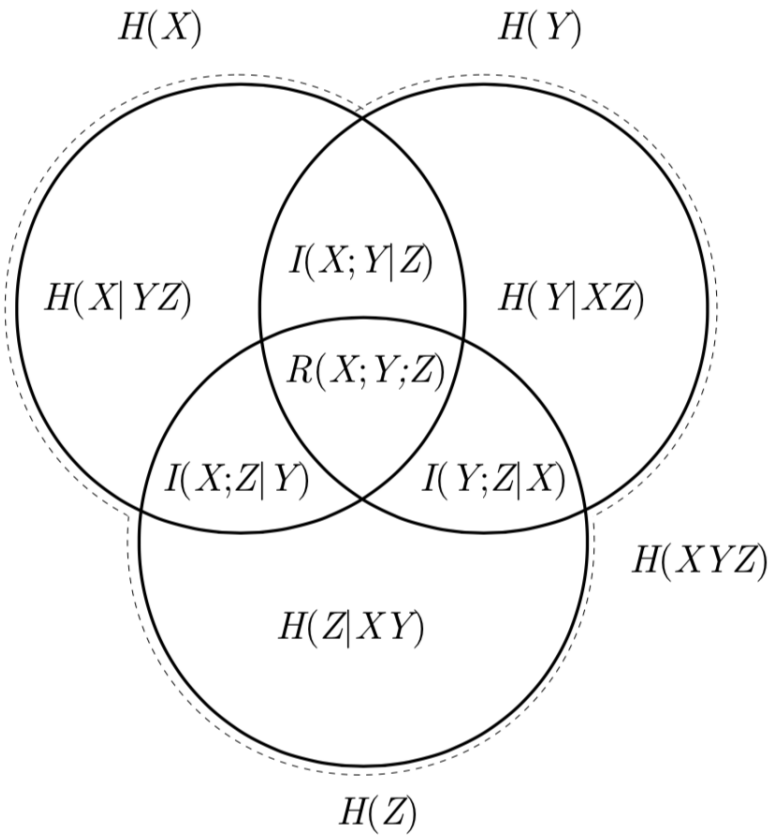
$$\begin{aligned} I(WX;Y|Z) &= H(WX|Z) - H(WX|YZ) \\ &= (H(X|Z) + H(W|XZ)) - (H(X|YZ) + H(W|XYZ)) \\ &= H(X|Z) - H(X|YZ) + H(W|XZ) - H(W|XYZ) \\ &= I(X;Y|Z) + I(W;Y|XZ). \end{aligned}$$

created: 2019-10-21

Entropy Diagrams for Three Random Variables

Just like the entropy diagrams for two random variables, we can visualize the relations between entropy, conditional entropy, mutual information, and conditional mutual information for three random variables. From the diagram, one can easily read off all the relations between the information-theoretic measures, like for instance $H(X|YZ) = H(X) - I(X;Z) - I(X;Y|Z)$, which is a relation that is otherwise not immediately obvious.

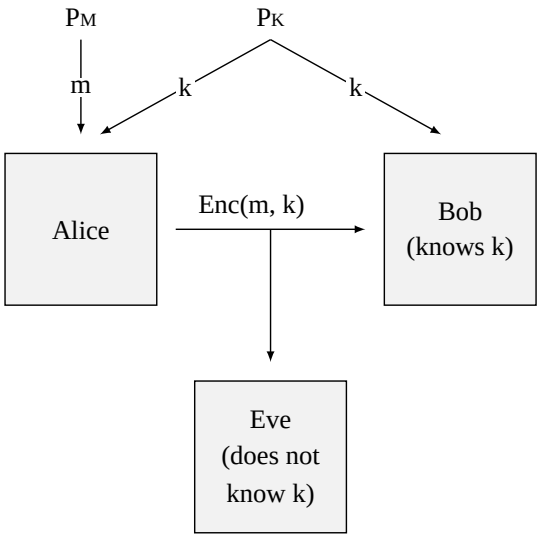
One subtlety with the entropy diagram for three random variables is that the "area in the middle'', $R(X;Y;Z) := I(X;Y) - I(X;Y|Z)$, may be *negative*. All other areas and quantities in the diagram are non-negative.



created: 2019-10-21

Definition: Perfectly Secure Encryption

Information theory is very useful when analyzing the security of perfectly secure encryption schemes. Consider a scenario where one party, Alice, wants to send a message m (sampled from some distribution P_M) to another party, Bob, over some public channel, for example the internet. Alice and Bob share a key k (sampled from another distribution P_K), which is a piece of information that is known only to them. Alice can use the key to encrypt her message (the **plaintext**), and Bob can use the same key to decrypt the **ciphertext** that Alice created, and read the message. The goal is to do this in such a way that if an eavesdropper (who usually goes by the name 'Eve') listens in on the channel and intercepts the encrypted message, she cannot derive any information about the message as long as she does not know the key k.



Let us formalize the above notion of encryption in the following definition:

Definition: Encryption scheme

An encryption scheme for (the message) M consists of a key K and a ciphertext $C = Enc(M, K)$, such that

- $I(M;K) = 0$ (the key is independent of the message -- this is a **setup assumption**), and

- $H(M|KC) = 0$ (given the key and the ciphertext, Bob can recover the original message)

Note that M, K and C are random variables.

Note that in order to satisfy the second requirement, the encryption function $Enc(\cdot, \cdot)$ needs to be injective: every message is mapped to a *unique* ciphertext.

The above definition does not put any constraints on the amount of information that Eve can get from the ciphertext: we still need to explicitly require the scheme to be secure.

Definition: Perfect security

An encryption scheme is perfectly secure if

$$I(M;C) = 0.$$

This is equivalent to saying $H(M|C) = H(M)$, or to saying that M and C are independent.

This type of security is also sometimes called perfect **information-theoric security**, in order to stress that the ciphertext really does not contain *any* information about the plaintext message. Many commonly used encryption schemes do not provide this type of security. In **computationally secure** schemes, a lot of information about the message may be contained in the ciphertext, but it would take a ridiculous amount of resources (such as computation time or memory) to compute the information about the message from the ciphertext.

One-Time Pad

A classic example of a perfectly secure encryption scheme is the one-time pad.

Definition: One-time pad (OTP)

Let the message space $\mathcal{M}$ be some additive group $(G, +)$. Define the random variable K to be uniformly distributed over the key space $\mathcal{K} = \mathcal{M}$, and define the ciphertext space to be $\mathcal{C} = \mathcal{M}$ as well. Define the encryption and decryption function as follows:

$$\begin{aligned} Enc(m, k) &= m + k = c, \\ Dec(c, k) &= c - k = m. \end{aligned}$$

Here, $c - k$ stands for $c + (-k)$, where $-k$ is the additive inverse of k in the group $(G, +)$.

Example: One-time pad for binary strings

The most common use of the one-time pad is for the group of binary strings under (bit-wise) addition modulo 2, i.e. $(\{0,1\}^n, \oplus)$. In this group, every element is its own additive inverse, resulting in the encryption and decryption functions

$$\begin{aligned} Enc(m, k) &= m \oplus k = c, \\ Dec(c, k) &= c \oplus k = m. \end{aligned}$$

For example, if $n = 4$, a possible message m is 0101, and a possible key k is 0110. The ciphertext c is $0101 \oplus 0110 = 0011$, and the decryption of c is again $0011 \oplus 0110 = 0101$, the original message m.

We can show that the one-time pad indeed satisfies the definition of perfect security.

Theorem

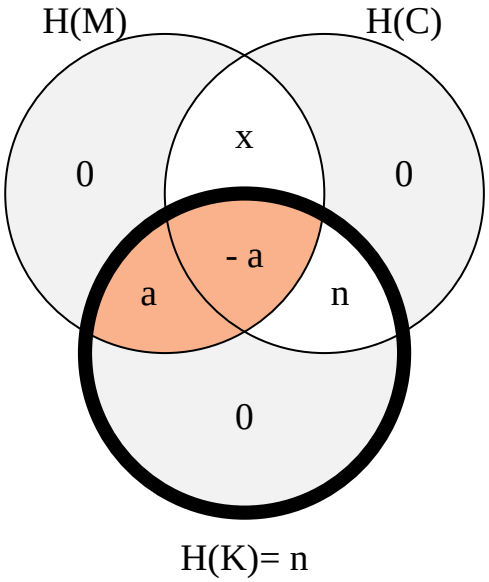
The one-time pad is perfectly secure.

Proof hint

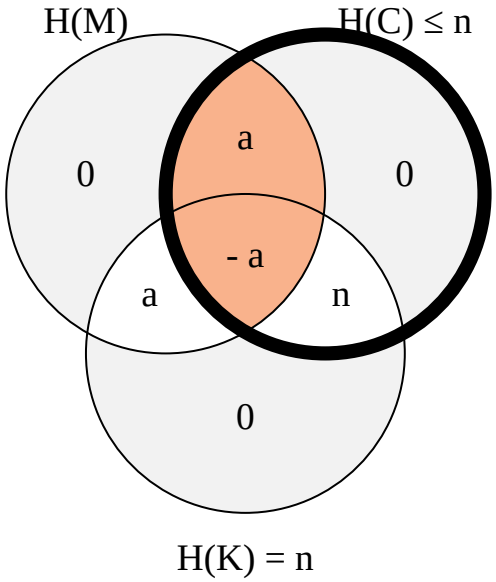
Draw a three-variable entropy diagram for the random variables M, K, and C. Use the fact that the key$K$ is picked uniformly at random, and the setup assumption that it is independent from the message M. Then deduce from the diagram that $I(M; C) = 0$, i.e., the message and ciphertext share no information.

Show full proof

Write $n = \log|G|$. We need to verify that $I(M;C) = 0$. We do so using a three-variable entropy diagram. We can already fill in the values $H(K) = n = \log|G|$ (because K is uniformly distributed), $H(M|CK) = H(C|MK) = H(K|MC) = 0$ (because each random variable is a function of the other two), and $I(M;K) = 0$ (this is our setup assumption).



Note that the area of $I(M;K) = I(M;K|C) + R(M;K;C)$ (shaded orange in the picture) as a whole is 0, but that does not mean that $I(M;K|C)$ and $R(M;K;C)$ themselves are zero, because $R(M;K;C)$ can be negative. We can conclude that there must be some (non-negative) real number $a \geq 0$ such that $I(M;K|C) = a$ and $R(M;K;C) = -a$. As the entropy of K has to be $H(K) = n$, we can furthermore conclude that $I(K;C|M) = n$. From $I(M;C) \geq 0$ follows that $x \geq a$, and because $H(C) \leq n$, it follows that $x \leq a$ and hence, $x = a$ and $I(M;C) = 0$, as desired.



We have thus seen that the one-time pad provides perfect information-theoretic security. There is one enormous drawback to this encryption scheme though: the key needs to be as large as the message! To send a message of n bits, Alice needs to share n bits of key with Bob. It might be tempting for Alice to reuse the key k

for several messages once she has shared it with Bob, but this is dangerous: Eve could, from two intercepted encryptions $(m_1 + k)$ and $(m_2 + k)$, recover the difference of the two plaintext messages $m_1 + k - (m_2 + k) = m_1 - m_2$. Already the difference between two plaintext messages can reveal a lot of information about the individual messages, as illustrated in this Cryptosmith blog post.

Minimum Key Length for Perfectly Secure Encryption

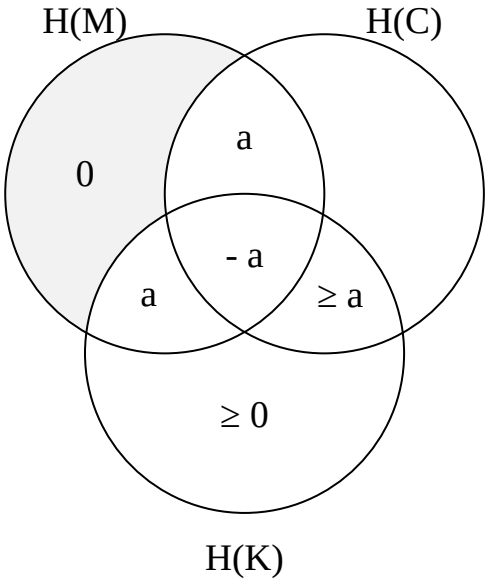
It turns out to be impossible to design an encryption scheme that provides both perfect security and short keys. So even though the one-time pad may seem inefficient, its key length is optimal for a perfectly secure scheme.

Theorem (Shannon, 1949)

For any perfectly secure encryption scheme, it holds that $H(K) \geq H(M)$.

Proof

Again, we turn to entropy diagrams. Write $a = I(M;C|K) \geq 0$. using the fact that $I(M;K) = 0$ (setup assumption) and $I(M;C) = 0$ (security), we can fill in the entropy diagram as follows:



Note that $I(K;C|M) \geq a$ follows from the fact that $I(C;K) \geq 0$ and $R(C;K;M) = -a$. From the diagram, we observe that $H(M) = a$, and that $H(K) \geq a$. Hence, $H(K) \geq H(M)$.

Introduction to Module 05

Previous modules:

- 01: Preliminaries (probability theory)
- 02: Building blocks (entropy, conditional entropy, mututal information, and relative entropy)
- 03/04: *compressing* information: encoding sources symbol-by-symbol (Huffman codes, arithmetic codes, Shannon's source coding theorem about the average codeword length of optimal codes), and encoding sources in blocks (typical sets).
- 04: *hiding* information: perfectly secure encryption (one-time pad).

So far in this course we have mostly seen independent random variables. In source coding, we designed our codes according to a single distribution P_X, and assumed that if we encoded a sequence of source symbols, the symbols in the sequence would all be drawn independently according to P_X. In the real world, however, subsequent events are often dependent on each other. For example, in an English text, after observing a letter `q`, the next letter is much more likely to be `u` than it is to be `r`, even though in general the letter `r` is more prevalent in English text. The event of observing the letter `q` changes the probability distribution of the next letter.

In this module, we study so-called *stochastic processes*, sequences of random variables which may depend on each other in various ways. We start by studying a restricted form of dependence between variables, Markov chains, and observe how information propagates through the sequence of random variables. We then move on to more general stochastic processes, and categorize them according to all kinds of properties they may have.

We end the module by quantifying how much information is contained in a random process. Because the samples in such a process are not necessarily independent, it does not make sense to measure their information content by the entropy of a single random variable. Instead, we define the entropy rate.

As an additional resource, you may find these student-recorded lectures useful. However, please remember that the course content is defined by the material

available on Canvas, so you may not get all the content by only watching the lecture recordings.

Markov Chains of Length 3

Random variables form a Markov chain if the distribution of each random variable depends only on the outcome of the random variable that directly precedes it.

Definition: Markov chain (of length 3)

The random variables X, Y, and Z form a Markov chain (notation: $X \to Y \to Z$) if and only if

$$P_{Z|XY} = P_{Z|Y}.$$

In this chapter, we regularly use the shortcut $P_{Z|XY} = P_{Z|Y}$ to denote that this equality should hold for all possible input values, i.e.

$$P_{Z|XY} = P_{Z|Y} \qquad \Leftrightarrow \qquad \forall x \in \mathcal{X}, y \in \mathcal{Y}, z \in \mathcal{Z} : P_{Z|XY}(z|x,y) = P_{Z|Y}(z|y)\,.$$

Proposition

For all random variables X, Y, Z, the following statements are equivalent:

a. $X \to Y \to Z$,
b. $Z \to Y \to X$,
c. $P_{XZ|Y} = P_{X|Y} \cdot P_{Z|Y}$ (that is, $I(X;Z|Y) = 0$).

Proof

We prove that (a) $\Rightarrow$ (b), that (b) $\Rightarrow$ (c), and that (c) $\Rightarrow$ (a). All other directions follow from (combinations of) these three implications.

- [(a) $\Rightarrow$ (b):] Suppose $X \to Y \to Z$. Then

$$\begin{aligned} P_{X|YZ} &= \frac{P_{XYZ}}{P_{YZ}} = \frac{P_{XY} \cdot P_{Z|XY}}{P_{YZ}} = \frac{P_{XY} \cdot P_{Z|XY}}{P_Y \cdot P_{Z|Y}} \\ &= \frac{P_{XY} \cdot P_{Z|Y}}{P_Y \cdot P_{Z|Y}} = \frac{P_{XY}}{P_Y} = P_{X|Y}\,, \end{aligned}$$

  where we go from the top to the bottom line using the assumption that $X \to Y \to Z$.
- [(b) $\Rightarrow$ (c):] Suppose $Z \to Y \to X$. Then

$$P_{XZ|Y} = \frac{P_{XYZ}}{P_Y} = \frac{P_{X|YZ} \cdot P_{YZ}}{P_Y}$$
$$= \frac{P_{X|Y} \cdot P_{YZ}}{P_Y} = P_{X|Y} \cdot \frac{P_{YZ}}{P_Y} = P_{X|Y} \cdot P_{Z|Y},$$

where we go from the top to the bottom line using the assumption that $Z \to Y \to X$.

- [(c) $\Rightarrow$ (a):] For the final implication, suppose that $P_{XZ|Y} = P_{X|Y} \cdot P_{Z|Y}$. Then

$$P_{Z|YX} = \frac{P_{XYZ}}{P_{XY}} = \frac{P_{XZ|Y} \cdot P_Y}{P_{XY}}$$
$$= \frac{P_{X|Y} \cdot P_{Z|Y} \cdot P_Y}{P_{XY}} = \frac{P_{XY} \cdot P_{Z|Y}}{P_{XY}} = P_{Z|Y},$$

and therefore $X \to Y \to Z$.

Because of the equivalence of items (a) and (b), we often write $X \leftrightarrow Y \leftrightarrow Z$ to denote a Markov chain.

Example: Whispering game

Alice, Bob and Charlie play a game: Alice comes up with a message, and whispers it into Bob's ear. Bob then proceeds to whisper the message into Charlie's ear, who says it out loud. Of course, Bob and Charlie may not hear the message correctly: there is some noise in the communication. Let A be the random variable that contains the message that Alice picks. For simplicity, let's say she picks a uniformly random bit. Let B be the message that Bob heard: it is also a bit, but with probability 0.1, it is not the same as A. More formally,

$$P_{B|A}(0|0) = P_{B|A}(1|1) = 0.9,$$
$$P_{B|A}(1|0) = P_{B|A}(0|1) = 0.1.$$

Finally, let C be the message that Charlie heard: again, let us suppose that it is unequal to the whispered message B with probability 0.1. For the conditional distribution of C, we see that $P_{C|AB}(0|00) = 0.9 = P_{C|B}(0|0)$, and similarly for all other possible values of A, B, and C. Thus, these random variables form a Markov chain: $A \to B \to C$. The proposition above tells us that we can also look at this game in the converse direction: if we are curious about Alice's original message A, we only have to look at Bob's interpretation B of the message. Additionally knowing Charlie's interpretation C does not change the distribution of A.

Data-Processing Inequality

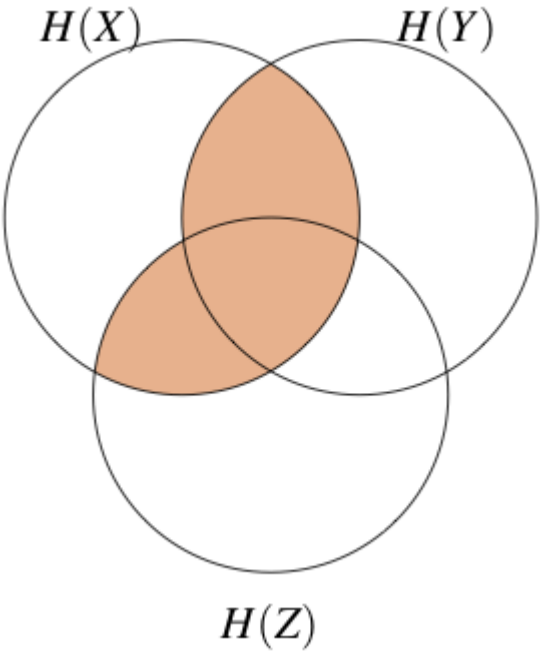
The whispering game in the example on the previous page exhibits an important property of Markov chains: you can only lose information down the line. Charlie's final message C does not contain any more information about Alice's original message A than what was already contained in Bob's message B. This observation is formalized in the following theorem:

Theorem: Data-processing inequality

If $X \to Y \to Z$, then $I(X;Y) \geq I(X;Z)$. Equality holds if and only if $I(X;Y|Z) = 0$.

Proof

The following entropy diagram depicts the area $I(X;YZ)$:



From the diagram, we can see that

$$I(X;Z) + I(X;Y|Z) = I(X;YZ) = I(X;Y) + I(X;Z|Y).$$

Combining this with part (c) of the proposition on the last page, it follows that

$$I(X;Z) + I(X;Y|Z) = I(X;Y).$$

Since $I(X;Y|Z) \geq 0$, the result follows: $I(X;Z) \leq I(X;Y)$, with equality iff $I(X;Y|Z) = 0$.
The following corollary formalizes the intuition that the mutual information between two random variables can only decrease by post-processing any of the

two.

Corollary

$I(X;Y) \geq I(X;g(Y))$ for any two random variables X and Y, and any function g on the range of Y.
Paste proof here

Sufficient Statistics

Consider a family of probability distributions P_X^θ which is parametrized by θ. Let $T(X)$ be any statistic (i.e. a function of sample X). It then holds that

$$\theta \to X \to T(X).$$

Hence, by the data-processing inequality, it holds that $I(\theta; X) \geq I(\theta; T(X))$, with equality if $I(\theta; X \mid T(X)) = 0$, or in other words, equality holds if $\theta \leftrightarrow T(X) \leftrightarrow X$ is also a Markov chain. As we want to make sure that our statistic does not lose any information about the parameter θ, we define the following.

Definition: Sufficient statistic

T(X) is a sufficient statistic if $P_{X|T(X)}$ is independent of θ for any distribution of θ.

Example: Coin Flips

Let $X_1, X_2, \ldots, X_n$ be iid coinflips, i.e. Bernoulli(p) variables and let $T(X_1, X_2, \ldots, X_n) = \sum_{i=1}^n X_i$ be a statistic. We have that

$$p \to X_1, \ldots, X_n \to \sum_{i=1}^n X_i = T(X_1, \ldots, X_n).$$

The probability of a particular outcome $x_1 \ldots x_n$ is given by

$$P_{X_1 \ldots X_n}(x_1, \ldots, x_n) = p^{T(x)} (1-p)^{n-T(x)}.$$

Observe that given that the number of 1's is $T(x)$, all strings with that property are evenly likely (and therefore independent of p). Hence $T(X) = \sum_{i=1}^n X_i$ is a sufficient statistic according to the definition above.

created: 2019-10-21

Definition: Longer Markov Chains

We can extend the definition of Markov chains to more than three variables:

Definition: Markov chain (of length n)

The random variables $X_1, X_2, \ldots, X_n$ form a Markov chain (notation: $X_1 \to X_2 \to \cdots \to X_n$) if for all $3 \leq i \leq n$,

$$P_{X_i|X_1 \cdots X_{i-1}} = P_{X_i|X_{i-1}} \, .$$

Markov chains of length n exhibit similar properties to the properties we have seen for Markov chains of length 3. In particular, the reverse chain is also a Markov chain ($X_n \to \cdots \to X_2 \to X_1$), and a more general form of the data-processing inequality holds in the sense that the further apart two variables are in the chain, the less correlated they are.

Proposition

If $X_1 \to X_2 \to X_3 \to X_4$ is a Markov chain, the following are Markov chains as well:

a. $X_1 \to X_2 \to X_3$
b. $X_2 \to X_3 \to X_4$
c. $X_1 \to X_2 X_3 \to X_4$
d. $X_4 \to X_3 \to X_2 \to X_1$

Proof

left as exercise

In the exercises, we will prove that if $X_1, X_2, \ldots, X_n$ forms a Markov chain, then for all $1 \leq i \leq j \leq k \leq n$: $I(X_i, X_j) \geq I(X_i, X_k)$. This is a generalized form of the data-processing inequality.

Discrete-Time Stochastic Process

Let us now consider more general sets of random variables than the three-variable Markov chains we saw in the previous section. In the rest of this chapter, we will consider infinite sequences of random variables, that can have varying degrees of independence. We start off with the most general definition of such an infinite process.

Definition: Discrete-time stochastic process

A stochastic process is a sequence $\{X_i : \Omega \to \mathcal{X}\}$ of random variables indexed by $i \in \mathbb{N}_+$. The process is characterized by the collection of probability distributions $P_{X_n|X_1 \cdots X_{n-1}}$ for all $n \in \mathbb{N}_+$.

Equivalently, we can say that a stochastic process is characterized by the joint probability distributions $P_{X_1 \cdots X_n}$ for all $n \in \mathbb{N}$. Note that the random variables all have the same domain (the same sample space Ω) and the same codomain $\mathcal{X}$. Often, this sample space is infinite, as it is in the following example.

Example: Repeatedly tossing a fair coin

Suppose you toss a fair coin an infinite amount of times, and at every step, you count the number of heads you have seen so far. This experiment can be described as a stochastic process by letting each variable X_i denote the number of heads observed up until that toss. For example, a sequence of tosses $\texttt{THHTHT}\cdots$ results in the values $X_1 = 0$, $X_2 = 1$, $X_3 = 2$, $X_4 = 2$, $X_5 = 3$, $X_6 = 3, \ldots$. The stochastic process is characterized by the probability distributions

$$P_{X_1}(0) = P_{X_1}(1) = \frac{1}{2} \qquad (P_{X_1}(x_1) = 0 \text{ for all other } x_1 \in \mathbb{N})$$

$$P_{X_{n+1}|X_1 \cdots X_n}(x_{n+1}|x_1 \cdots x_n) = \frac{1}{2} \text{ if } x_{n+1} = x_n \text{ or } x_{n+1} = x_n + 1, \text{ and } 0 \text{ otherwise.}$$

For this experiment, the sample space is the set of all possible outcomes for an infinite sequence of coin tosses,

$$\Omega = \{\texttt{H}, \texttt{T}\} \times \{\texttt{H}, \texttt{T}\} \times \{\texttt{H}, \texttt{T}\} \times \cdots$$

In particular, $|\Omega|$ is uncountable. Properly defining a probability measure for Ω would require us to rethink quite a few of our definitions that work only for finite sample spaces. That can be done (by using σ-**algebras** as event spaces), but that is beyond the scope of this course. For stochastic processes, we can always think of the sample space for X_n as a finite set (consisting of the first n samples only). In the current example, the probability distribution of X_n only depends on the first n coin tosses.

created: 2019-10-21

Stationary Process

Often, we will be interested in stochastic processes with specific properties, such as processes where all X_i are independent, or processes with a Markov-like property. We review several such properties.

Definition: Stationary process

A stochastic process is stationary if for all $n, k \in \mathbb{N}_+$,

$$P_{X_1 \cdots X_n} = P_{X_{1+k} \cdots X_{n+k}}.$$

Stationary processes are invariant under time shifts: when observing a subsequence of length n, it does not matter where in the process you look exactly.

Example: i.i.d. process

Let X be a random variable. Consider a stochastic process $\{X_i\}$ where $P_{X_i} = P_X$ for all i. That is, the random variables in the sequence are all independent and identically distributed. This process is stationary, since for any n, k, it holds that

$$P_{X_1 \cdots X_n} = \prod_{i=1}^{n} P_{X_i} = \prod_{i=1+k}^{n+k} P_{X_i} = P_{X_{1+k} \cdots X_{n+k}}.$$

Example: Ten fair coins

The following is another example of a stationary process. Throw a fair coin 10 times: this can be described by the finite sample space $\{\mathtt{H}, \mathtt{T}\}^{10}$. Define the stochastic process $\{X_i\}_{i \in \mathbb{N}_0}$ by setting

$$X_i = \begin{cases} 1 & \text{if the } [i \bmod 10]\text{th coin lands on heads} \\ 0 & \text{otherwise.} \end{cases}$$

Here, $[k \bmod N]$ is defined to be an element of $\{0, 1, 2, \ldots, N-1\}$. If we want the first variable in the process to be X_1 instead of X_0, we can determine the value of X_i based on the $[((i-1) \bmod 10) + 1]$th coin.

As an exercise, show that this process is indeed stationary.

Hint

Observe that for all i, $X_i = X_{i+10} = X_{i+20} = \ldots$

Markov Process: Time Invariance, Finite State, Transition Matrix

Definition: Markov process

A stochastic process is a Markov process (or: Markov chain) if for all $n \in \mathbb{N}$,

$$X_1 \to X_2 \to \cdots \to X_n.$$

In a Markov process, the value of each step can only depend on the previous value, similarly to the three-variable Markov chains from earlier. The infinite sequence of coin tosses from earlier is an example of a Markov process: you can see this by inspecting the definition of $P_{X_{n+1}|X_1 \cdots X_n}$, and noting that it is indeed independent of the values for X_1 to X_{n-1}. The process in this example even fulfills a stronger property: it is time invariant.

Definition: Time-invariant Markov process

A Markov process is time invariant if for all $n \in \mathbb{N}$, and for all $a, b \in \mathcal{X}$,

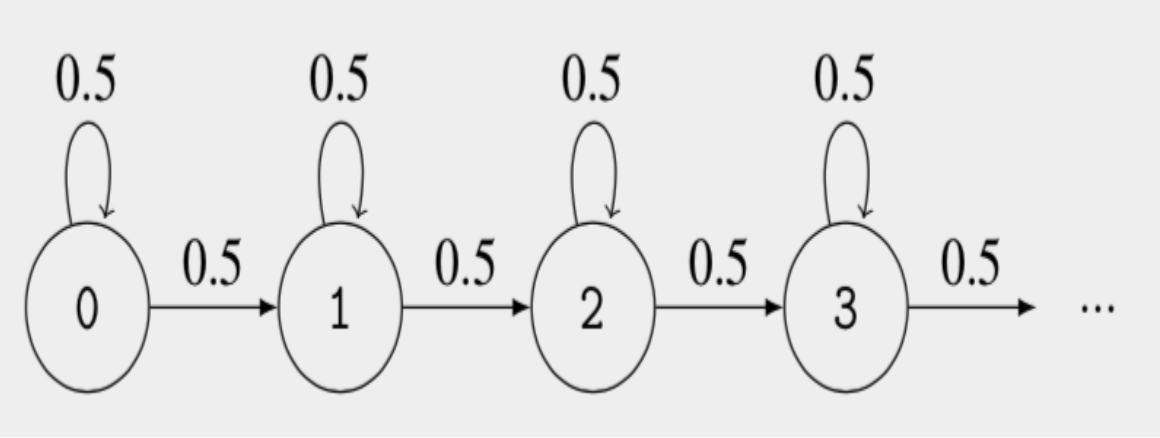
$$P_{X_{n+1}|X_n}(a|b) = P_{X_2|X_1}(a|b) \qquad \text{whenever } P_{X_n}(b) > 0 \text{ and } P_{X_1}(b) > 0.$$

Time-invariant Markov processes can be nicely visualized using state diagrams, where the **states** represent the values in $\mathcal{X}$, and the labels on the arrows represent the conditional probabilities. A time-dependent Markov process could also be visualized in this way, but the labels on the arrows would have to be functions of the time step i. In both cases, one also needs to specify P_{X_1}, the **initial distribution**, in order to completely describe the stochastic process.

Example 1: Repeatedly tossing a fair coin, continued

The infinite sequence of coin tosses is represented by the following state diagram:

created: 2019-10-21



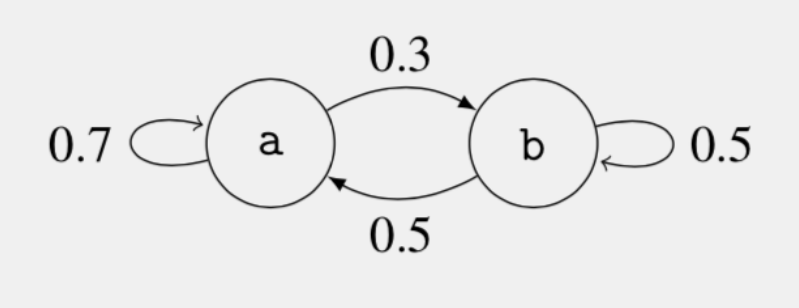
For example, the arrow from state 2 to state 3, represents the probability $P_{X_{n+1}|X_n}(3|2) = 0.5$. The initial distribution is $P_{X_1}(0) = P_{X_1}(1) = 0.5$: the process is equally likely to start out in state 0 (if the first toss is a tails) or state 1 (if the first toss is a heads).

Definition: Finite-state Markov process

A Markov process is finite-state if $|\mathcal{X}|$ is finite.

Example 2: A finite-state time-invariant Markov process

Consider the following state diagram, for a Markov process with initial distribution $P_{X_1}(\mathtt{a}) = 1$ and $P_{X_1}(\mathtt{b}) = 0$:



The process starts in state **a** with probability one. A possible run of the process would be **aabaaabbabaa**$\cdots$.

Note that in the state diagrams, the probabilities of the outgoing arrows add up to 1 for every state. This is necessary for a well-defined Markov process. A time-invariant Markov process can alternatively be represented by its initial distribution combined with a **transition matrix** R, where the entry R_{ij} represents the transition probability $P_{X_{n+1}|X_n}(j|i)$. For a finite-state process, the transition matrix is finite. In the transition matrix, the row entries have to sum up to 1. When the current state is given by a row vector v_n, the state after one step of the Markov process is given by $v_{n+1} = vR$. If you (like me) don't like multiplying vectors from the left with matrices, you can also compute $v_{n+1} = \left(R^T v^T\right)^T$ where $(\cdot)^T$ denotes the transposition of matrices and vectors.

Example 2: A finite-state time-invariant Markov process, continued

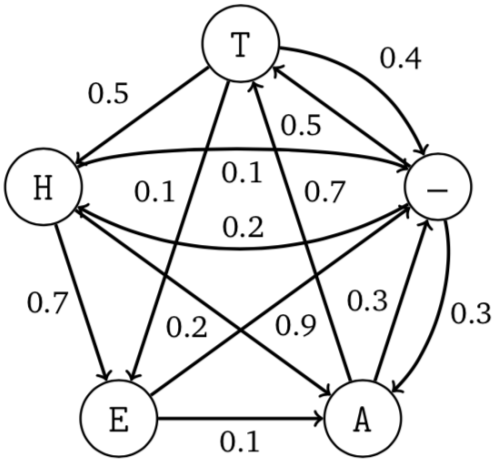
The matrix representation of the process above is given by

$$R = \begin{bmatrix} 0.7 & 0.3 \\ 0.5 & 0.5 \end{bmatrix}$$

where the state $\mathtt{a}$ is represented by the first row/column, and the state $\mathtt{b}$ by the second. Verify that the row entries indeed sum up to 1. The initial distribution is still given by $P_{X_1}(\mathtt{a}) = 1$ and $P_{X_1}(\mathtt{b}) = 0$.

Another example: primitive language model

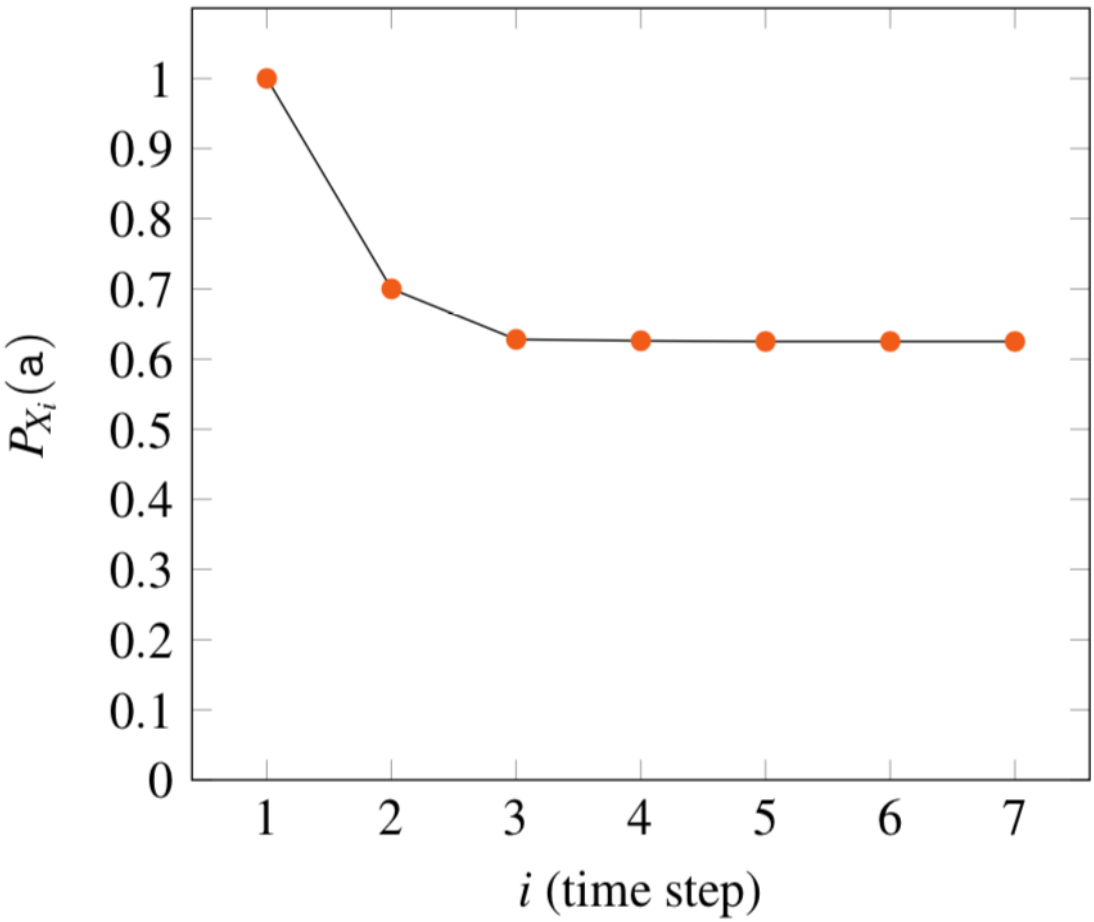
Observe how this very simple 5-state Markov chain produces samples that resemble English language. This is a first hint of how surprisingly powerful Markov models can be.




```
T_ATE_T_HE_TE_THE_THE_THAT_T_TE_
ATHE_AT_ATHE_T_ATHE_TE_ATH_TH_A_
A_THE_THE_THATEA_THE_HE_A_T_ ...
```

(image by Mathias Madsen)

Markov Process: Stationary Distribution



Suppose that we run the process of Example 2 for a very large number of steps, and wonder what the probability will be of observing an **a** at the next step. Given the initial distribution and the state diagram, we can compute the probability distribution for every X_i. In the figure above, $P_{X_i}(\mathtt{a})$ is plotted for several values of i. The probability to observe an **a** seems to stabilize. This leads us to the following definition:

Definition: Stationary distribution

A stationary distribution for a time-invariant Markov chain is a distribution P_{X_n} such that $P_{X_{n+1}} = P_{X_n}$.

created: 2019-10-21

If the initial distribution of a time-invariant Markov process is stationary, then the entire process is stationary as defined previously.

Proposition

Every time-invariant finite-state Markov process has a stationary distribution.

Proof

Let $k := |\mathcal{X}|$. The $k \times k$ transition matrix R with entries $R_{ij} = P_{X_{n+1}|X_n}(j|i)$ is a stochastic matrix, as for every row i, the sum over columns is $\sum_{j=1}^{k} R_{ij} = 1$. We are interested in finding a vector $v \in \mathbb{R}^k_{\geq 0}$ such that $\|v\| = 1$ and $R^T v = v$. This vector then represents the stationary distribution. Clearly, a possible eigenvector for R is the all-1 vector $w = (1, \ldots, 1)^T$ because $Rw = w$ by definition of a stochastic matrix. Hence, 1 is an eigenvalue of R. As R and R^T have the same eigenvalues, 1 is also an eigenvalue of R^T; let $v \in \mathbb{R}^k$ be the corresponding eigenvector such that $R^T v = v$. If all coordinates of v are non-negative, one can verify that we have found a stationary distribution by renormalizing $v / \sum_{i=1}^{k} v_i$. Otherwise, let us write $v = v^+ - v^-$ with $v^+, v^- \in \mathbb{R}^k_{\geq 0}$, where we put all positive coordinates of v in v^+ and all negative coordinates of v in v^-. Note that $R^T v^+ - R^T v^- = R^T(v^+ - v^-) = R^T v = v = v^+ - v^-$. As all entries of R^T, v^+ and v^- are positive, equality must hold for both the positive and negative parts: $R^T v^+ = v^+$ and $R^T v^- = v^-$. As either $v^+ \neq 0^k$ or $v^- \neq 0^k$ (otherwise $v = 0^k$, which cannot be the case for an eigenvector), renormalizing that non-zero vector as above yields the stationary distribution.

Given the transition matrix R of a finite-state Markov process, one can find the stationary distribution μ by solving the linear equation $\mu R = \mu$ under the constraint that $\sum_i \mu_i = 1$.

Example 2: A finite-state time-invariant Markov process, continued

The matrix representation of the process above is given by

$$R = \begin{bmatrix} 0.7 & 0.3 \\ 0.5 & 0.5 \end{bmatrix}$$

. Writing out $(\mu_a, \mu_b)R = (\mu_a, \mu_b)$ results in

$$\begin{cases} 0.7\mu_a + 0.5\mu_b = \mu_a \\ 0.3\mu_a + 0.5\mu_b = \mu_b \end{cases}$$

. These are linearly dependent equations, but together with the constraint $\mu_a + \mu_b = 1$, they can be solved to $(\mu_a, \mu_b) = (5/8, 3/8)$.

Markov Process: Irreducibility, Periodicity, Convergence

Every time-invariant Markov process has a stationary distribution, but it might not be unique, and the process might never reach the stationary distribution. For uniqueness and convergence, we need additional requirements on the Markov process.

Definition: Irreducible Markov process

A time-invariant Markov process is irreducible if every state is reachable from any other state in a finite number of steps.

The process in Example 2 is irreducible: the state **a** is reachable from **b** and vice versa. The process in Example 1 is not irreducible. For example, the state 0 is not reachable from the state 2.

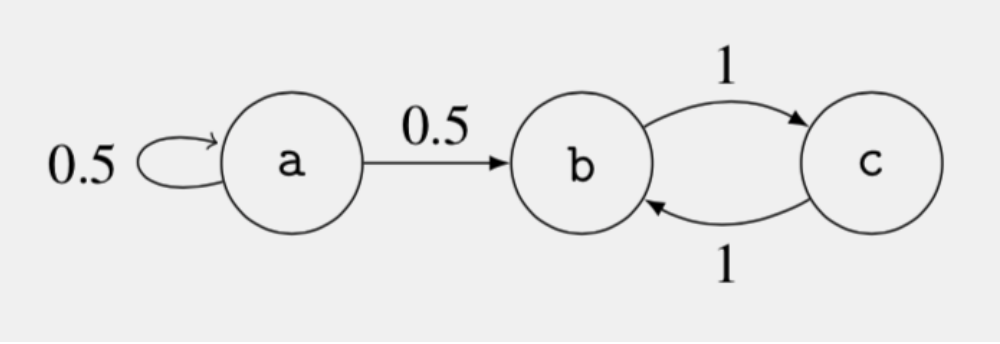
Definition: Aperiodic Markov process

A state in a time-invariant Markov process is aperiodic if the greatest common divisor of all path lengths from that state to itself is 1. The process itself is aperiodic if all states are aperiodic.

The processes in Example 1 and Example 2 are both aperiodic. In both examples, every state is reachable from itself with paths of any length, so the greatest common divisor of the path lengths is always 1. Below, we will present an example of a process that is periodic (i.e., not aperiodic).

Example 3: Periodic Markov process

Consider the process that starts in state **a** with probability 1, and is further described by the following state diagram:



The state **a** is aperiodic, but **b** and **c** are not. The paths from state **b** to itself are all

of even length, as are the paths from state **c** to itself. A typical run of this process might look like **aaaaabcbcbcbc**; the period 2 is clearly visible in this run.
decide whether this button is necessary, and whether the title of this block should be question instead of example

Proposition

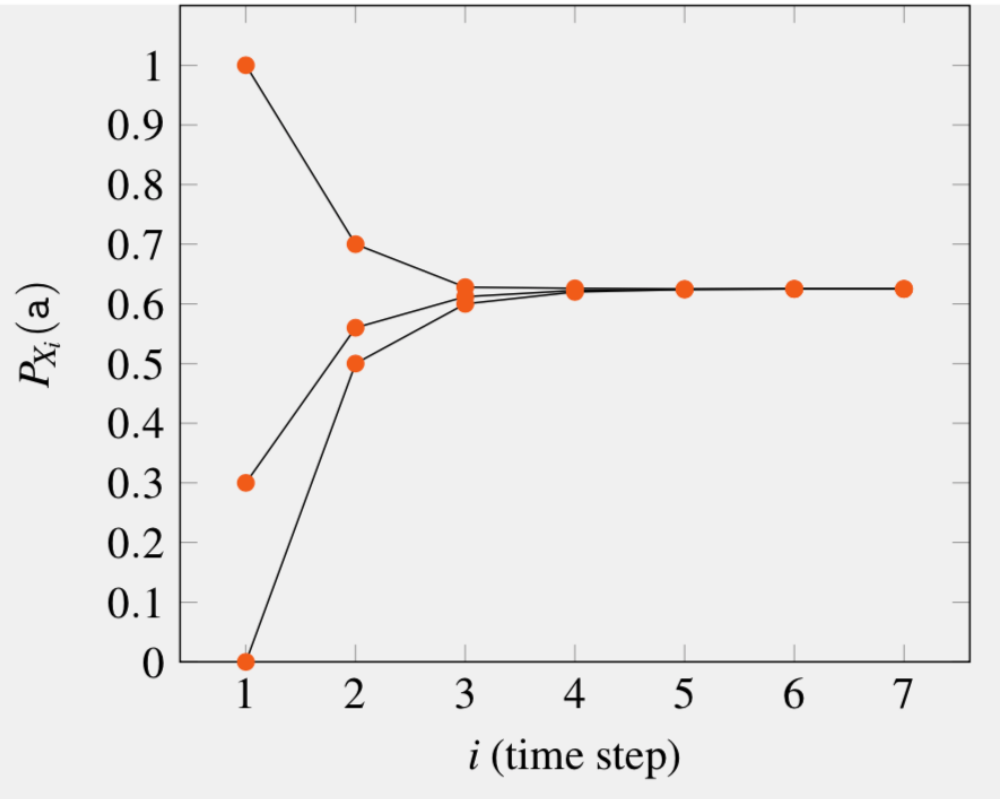
If a time-invariant Markov chain is finite-state, irreducible and aperiodic, then there exists a unique stationary distribution. Moreover, from any initial distribution, the distribution of X_n tends to the stationary distribution as $n \to \infty$.
Paste proof here

Example 2, continued

Consider again the process of Example 2. This process does have a stationary distribution, because it is finite-state, irreducible and aperiodic. Let $\mu_{\mathtt{a}}$ denote the probability of observing an **a** in the stationary distribution, and $\mu_{\mathtt{b}}$ the probability of observing a **b**. Then the stationary distribution must satisfy the following set of equations:

$$\begin{aligned} \mu_{\mathtt{a}} &= 0.7\mu_{\mathtt{a}} + 0.5\mu_{\mathtt{b}}, \\ \mu_{\mathtt{b}} &= 0.3\mu_{\mathtt{a}} + 0.5\mu_{\mathtt{b}}, \\ \mu_{\mathtt{a}} + \mu_{\mathtt{b}} &= 1. \end{aligned}$$

Solving this set of equations gives the stationary distribution $\mu_{\mathtt{a}} = 0.625$ and $\mu_{\mathtt{b}} = 0.375$. This outcome matches our observations in the figure below. The starting distribution is irrelevant, because in the limit, the stationary distribution is reached. The following plot exemplifies this for three different starting distributions ($P_{X_1}(\mathtt{a}) = 0$, $P_{X_1}(\mathtt{a}) = 0.3$, and $P_{X_1}(\mathtt{a}) = 1$).



Entropy Rate

Intuitively, some of the stochastic processes we have seen in the previous sections are more predictable than others. The periodic Markov process from Example 3 is not so surprising anymore as soon as the first b is observed. In this section, we will study a measure for the unpredictability of a stochastic process: the entropy rate. It is a natural analogue to the entropy of single/independent random variables.

Definition: Entropy rate

The entropy rate $H(\{X_i\})$ of a stochastic process $\{X_i\}$ is

$$H(\{X_i\}) := \lim_{n\to\infty} \frac{1}{n} H(X_1, \ldots, X_n),$$

if the limit exists, and undefined otherwise.

In the literature, the entropy rate is often denoted $H(\mathcal{X})$, referring to the common support of the variables in the stochastic process. The notation $H(X)$ is also sometimes used, but this can be ambiguous and confusing. The entropy rate reflects the way in which the entropy of the sequence (observed so far) grows as n grows large.

Example

Consider a process $\{X_i\}$ where the X_i are i.i.d. sampled from P_X. Then

$$\begin{aligned} H(\{X_i\}) &= \lim_{n\to\infty} \frac{1}{n} H(X_1, \ldots X_n) \\ &= \lim_{n\to\infty} \frac{1}{n} (H(X_1) + H(X_2) + \ldots + H(X_n)) \\ &= \lim_{n\to\infty} \frac{n}{n} H(X) \\ &= H(X). \end{aligned}$$

So, every new coin toss increases the entropy of the entire observed sequence by $H(X)$. This example shows that for i.i.d. processes, the entropy rate coincides with regular Shannon entropy.

We can also define an alternative measure of the unpredictability of a stochastic process, where we focus not on the amount of entropy in the entire sequence observed so far, but on the amount of entropy present in the current random variable, given the past sequence.

Definition: Entropy rate given the past

The entropy rate given the past $H'(\{X_i\})$ of a stochastic process $\{X_i\}$ is

$$H'(\{X_i\}) := \lim_{n\to\infty} H(X_n | X_1, \ldots, X_{n-1}),$$

if the limit exists, and undefined otherwise.

For all stationary processes, this alternative definition turns out to coincide with the original definition of entropy rate. In order to show this, we need an analytic statement about the convergence of sums.

Theorem: Cesàro mean

If $\lim_{n\to\infty} a_n = a$ and $b_n = \frac{1}{n}\sum_{i=1}^{n} a_i$, then $\lim_{n\to\infty} b_n = a$.

Proof

05 Cesàro mean.mp4

Theorem

For a stationary process $\{X_i\}$, it holds that $H(\{X_i\}) = H'(\{X_i\})$ (and both limits exist).

Proof

We first show that $H(X_n \mid X_1, \dots, X_{n-1})$ is a non-increasing function of n:

$$\begin{aligned} H(X_n|X_1,\dots,X_{n-1}) &= H(X_{n+1}|X_2,\dots,X_n) && \text{(stationary)} \\ &\geq H(X_{n+1}|X_1,X_2,\dots,X_n) && \text{(Bounds on the Conditional Entropy)}. \end{aligned}$$

Combined with the fact that $H(X_n \mid X_1, \dots, X_{n-1})$ is lower bounded by 0, this implies that the limit $\lim_{n\to\infty} H(X_n \mid X_1, \dots, X_{n-1})$ must exist. It is $H'(\{X_i\})$. It remains to show that $H(\{X_i\}) = H'(\{X_i\})$:

$$\begin{aligned} H(\{X_i\}) &= \lim_{n\to\infty} \frac{1}{n} H(X_1, \dots, X_n) \\ &= \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} H(X_i \mid X_1, \dots, X_{i-1}) \\ &= H'(\{X_i\}). \end{aligned}$$

The final equality follows from the Cesaro mean.

Example 2: A finite-state time-invariant Markov process, continued

For a Markov process with transition matrix

$$R = \begin{bmatrix} 0.7 & 0.3 \\ 0.5 & 0.5 \end{bmatrix},$$

we have previously computed the stationary distribution to be $(\mu_a, \mu_b) = (5/8, 3/8)$. Hence, if we start in this stationary distribution, the entropy rate for this time-invariant stationary Markov process is

$$H(\{X_i\}) = H'(\{X_i\}) = \lim_{n\to\infty} H(X_n|X^{n-1}) = H(X_2|X_1)\,,$$

where P_{X_1} is the stationary distribution, i.e. $P_{X_1}(\mathtt{a}) = 5/8, P_{X_1}(\mathtt{b}) = 3/8$. Therefore,

$$\begin{aligned} H(\{X_i\}) = H(X_2|X_1) &= P_{X_1}(\mathtt{a}) \cdot H(X_2|X_1 = \mathtt{a}) + P_{X_1}(\mathtt{b}) \cdot H(X_2|X_1 = \mathtt{b}) \\ &= 5/8 \cdot h(0.3) + 3/8 \cdot h(0.5) \\ &\approx 0.926\,. \end{aligned}$$

Note that the entropy rate $H(\{X_i\}$ is not equal to the entropy of the stationary distribution (which is $h(5/8) \approx 0.792$ in this case)!

Source-Coding Theorem for Stationary Stochastic Processes

Recall Shannon's Source-Coding Theorem: it states that an optimal code (for an i.i.d. source X) has expected codeword length approximately $H(X)$.

We can state a similar result for stochastic processes:

Theorem: Source-coding theorem (for stochastic processes)

Let $\{X_i\} = X_1, X_2, X_3, \dots$ be a stationary stochastic source. Let $\ell_{\min}(n)$ be the expected minimal codeword length *per symbol* when encoding blocks of n source symbols, that is, $\ell_{\min}(n) := \ell_{\min}(P_{X_1 \cdots X_n})/n$. Then

$$\lim_{n \to \infty} \ell_{\min}(n) = H(\{X_i\}).$$

Proof

By Shannon's souce-coding theorem, we have that for every n,

$$H(X_1 X_2 \cdots X_n) \leq \ell_{\min}(P_{X_1 X_2 \cdots X_n}) \leq H(X_1 X_2 \cdots X_n) + 1.$$

Dividing all sides by n, and recalling that for stationary processes, $H(X_1 X_2 \cdots X_n)/n$ converges to the entropy rate $H(\{X_i\})$, the result follows.

In the limit, we can compress a stationary stochastic source down to its entropy rate.

created: 2019-10-21

AEP for Ergodic Stationary Processes

We just saw that Shannon's source coding theorem generalizes from i.i.d. sources to stationary stochastic sources. Under some natural assumptions, many statements and interpretations of the Shannon entropy we have seen in the context of the asymptotic equipartition property (AEP) can be generalized to the entropy rate $H(\{X_i\})$ of stochastic processes. We do require that the process behaves naturally in the following sense:

Definition: Ergodic Random Processes

A stochastic process $\{X_i\}$ is **ergodic** if its statistical properties can be deduced from a single, sufficiently long, random sample of the process.

There are many equivalent ways of giving precise mathematical definitions of this notion, but this goes beyond the scope of this course. Instead, we consider the following examples.

Example

Suppose we repeatedly pick a letter at random and print it three times:
`LLL EEE HHH QQQ MMM QQQ OOO TTT EEE YYY XXX GGG...`

This random process is ergodic (as we see all letters eventually) but not stationary.

On the other hand, if we pick a letter at random and print it forever:
`GGGGGGGGGGGGGGGGGGG...`

This process is stationary, but not ergodic (from one sample of the process, we can only see a single letter).

Shannon–McMillan–Breiman theorem: AEP

If $H(\{X_i\})$ is the entropy rate of a finite-valued stationary ergodic process $\{X_i\}$, then

$$-\frac{1}{n}\log P_{X_1,\dots X_n}(X_1,\dots X_n) \to H(\{X_i\}) \qquad \text{with probability } 1.$$

created: 2019-10-21

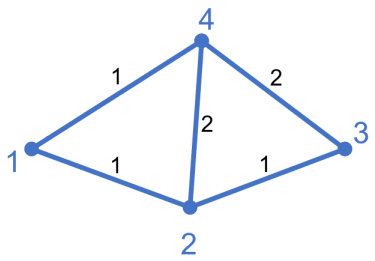
In other words, the asymptotic equipartition property holds for such processes: we can define typical sets and all the results about data compression we have seen in the previous module not only hold for iid processes, but more generally for stationary ergodic processes.

created: 2019-10-21

Random Walks on Graphs

An important and widely applicable example of a time-invariant Markov process is a random walk on a connected graph G with strictly positive symmetric edge weights $W_{ij} = W_{ji}$. The random walk is defined as follows: at node i, walk to node j with probability $\frac{W_{ij}}{W_i}$ where $W_i := \sum_j W_{ij}$ is the sum of the weights of all edges involving node i, and $W := \frac{1}{2}\sum_i W_i$ is the total of all edge weights.

Example



For the following graph , we have that $W_1 = 2, W_2 = 4, W_3 = 3, W_4 = 5$ and $2 \cdot W = \sum_i W_i = 2 \cdot 7$.

The stationary distribution of this random walk is given by $\mu_i := \frac{W_i}{2W}$, because indeed, at every node i, we have that the sum of all incoming weight is

$$\sum_j \mu_j \frac{W_{ij}}{W_j} = \sum_j \frac{W_j}{2W}\frac{W_{ij}}{W_j} = \frac{W_i}{2W} = \mu_i \,.$$

We continue to compute the entropy rate of this random walk. Assuming we start in the stationary distribution, we can compute the entropy rate as follows.

$$\begin{aligned}
H(\{X_i\}) = \sum_i \mu_i H(\ldots \frac{W_{ij}}{W_i} \ldots) &= -\sum_i \mu_i \sum_j \frac{W_{ij}}{W_i} \log \frac{W_{ij}}{W_i} \\
&= -\sum_{i,j} \frac{W_i}{2W} \cdot \frac{W_{ij}}{W_i} \log\left(\frac{W_{ij}}{2W} \cdot \frac{2W}{W_i}\right) \\
&= -\sum_{i,j} \frac{W_{ij}}{2W} \log\left(\frac{W_{ij}}{2W}\right) + \sum_{i,j} \frac{W_{ij}}{2W} \log\left(\frac{W_i}{2W}\right) \\
&= -\sum_{i,j} \frac{W_{ij}}{2W} \log\left(\frac{W_{ij}}{2W}\right) + \sum_{i} \frac{W_i}{2W} \log\left(\frac{W_i}{2W}\right) \\
&= H(\ldots \frac{W_{ij}}{2W} \ldots) - H(\ldots \frac{W_i}{2W} \ldots)
\end{aligned}$$

which is the difference of the entropy of the edge distribution and the entropy of the stationary distribution.

Example, continued

In the example above, the edge distribution is $\frac{1}{14}(1,1,1,2,2,1,1,1,2,2)$ and the stationary distribution is $\frac{1}{14}(2,4,3,5)$, resulting in
$H(\{X_i\}) = H(\frac{1}{14}(1,1,1,2,2,1,1,1,2,2)) - H(\frac{1}{14}(2,4,3,5)) \approx 1.312$

2019: Hidden Markov Models

\[\begin{align}

\sum_{k=(n+1)/2}{n} \binom{n}{k} 0.1^k 0.9^{n-k} &\approx \binom{n}{(n+1)/2} 0.1^{(n+1)/2} 0.9^{(n-1)/2}\

&= \binom{n}{(n+1)/2} 0.1 \cdot 0.09^{(n-1)/2}\

&\approx 2^{n \cdot h(1/2)} \cdot 0.1 \cdot 0.09 \cdot {(n-1)/2}\

&= 2^{n} \cdot 0.1 \cdot 0.09 \cdot {(n-1)/2}\

\end{align}]

Intuitively, some of the stochastic processes we have seen in the previous sections are more predictable than others. The periodic Markov process from Example 3 is not so surprising anymore as soon as the first b is observed. In this section, we will study a measure for the unpredictability of a stochastic process: the entropy rate.

Definition: Entropy rate

The entropy rate $H(\{X_i\})$ of a stochastic process $\{X_i\}$ is

$$H(\{X_i\}) := \lim_{n\to\infty} \frac{1}{n} H(X_1, \ldots, X_n),$$

if the limit exists, and undefined otherwise.

In the literature, the entropy rate is often denoted $H(\mathcal{X})$, referring to the common support of the variables in the stochastic process. The notation $H(X)$ is also sometimes used, but this can be ambiguous and confusing. The entropy rate reflects the way in which the entropy of the sequence (observed so far) grows as n grows large.

Example

Consider a process $\{X_i\}$ where the X_i are i.i.d. sampled from P_X. Then

$$\begin{aligned} H(\{X_i\}) &= \lim_{n\to\infty} \frac{1}{n} H(X_1, \ldots X_n) \\ &= \lim_{n\to\infty} \frac{1}{n} (H(X_1) + H(X_2) + \ldots + H(X_n)) \\ &= \lim_{n\to\infty} \frac{n}{n} H(X) \\ &= H(X). \end{aligned}$$

So, every new coin toss increases the entropy of the entire observed sequence by $H(X)$.

Show solution

decide whether this button is necessary, and whether the title of this block should be question instead of example

We can also define an alternative measure of the unpredictability of a stochastic process, where we focus not on the amount of entropy in the entire sequence observed so far, but on the amount of entropy present in the current random variable, given the past sequence.

Definition: Entropy rate given the past

The entropy rate given the past $H'(\{X_i\})$ of a stochastic process $\{X_i\}$ is

$$H'(\{X_i\}) := \lim_{n\to\infty} H(X_n|X_1,\dots,X_{n-1}),$$

if the limit exists, and undefined otherwise.

For all stationary processes, this alternative definition turns out to coincide with the original definition of entropy rate. In order to show this, we need an analytic statement about the convergence of sums.

Theorem: Cesàro mean

If $\lim_{n\to\infty} a_n = a$ and $b_n = \frac{1}{n}\sum_{i=1}^{n} a_i$, then $\lim_{n\to\infty} b_n = a$.

Proof

05 Cesàro mean.mp4

Theorem

For a stationary process $\{X_i\}$, it holds that $H(\{X_i\}) = H'(\{X_i\})$ (and both limits exist).

Proof

We first show that $H(X_n \mid X_1,\dots,X_{n-1})$ is a non-increasing function of n:

$$\begin{aligned} H(X_n|X_1,\dots,X_{n-1}) &= H(X_{n+1}|X_2,\dots,X_n) && \text{(stationary)} \\ &\geq H(X_{n+1}|X_1,X_2,\dots,X_n) && \text{(Bounds on the Conditional Entropy)}. \end{aligned}$$

Combined with the fact that $H(X_n \mid X_1,\dots,X_{n-1})$ is lower bounded by 0, this implies that the limit $\lim_{n\to\infty} H(X_n \mid X_1,\dots,X_{n-1})$ must exist. It is $H'(\{X_i\})$. It remains to show that $H(\{X_i\}) = H'(\{X_i\})$:

$$\begin{aligned} H(\{X_i\}) &= \lim_{n\to\infty} \frac{1}{n} H(X_1,\dots,X_n) \\ &= \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} H(X_i \mid X_1,\dots,X_{i-1}) \\ &= H'(\{X_i\}). \end{aligned}$$

The final equality follows from the Cesaro mean.

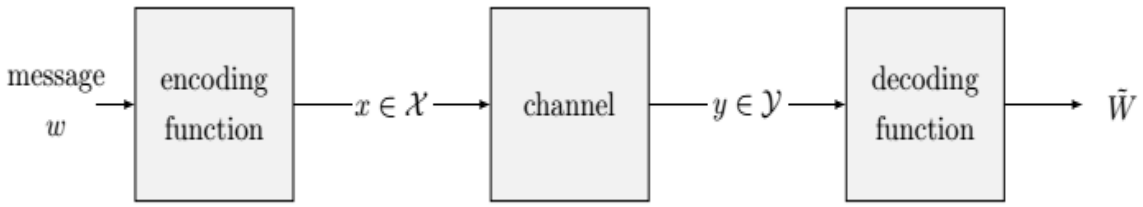
Introduction to Module 06

Previous modules:

- 01: Preliminaries (probability theory)
- 02: Building blocks (entropy, conditional entropy, mututal information, and relative entropy)
- 03/04: *compressing* information: encoding sources symbol-by-symbol (Huffman codes, arithmetic codes, Shannon's source coding theorem about the average codeword length of optimal codes), and encoding sources in blocks (typical sets).
- 04: *hiding* information: perfectly secure encryption (one-time pad).
- 05: generalization to stochastic processes (entropy rate)

In Module 03, we saw how to encode information from a source to compress it, so that it can be stored or sent as efficiently as possible. While compression is great for efficiency, there is also a danger to it: because the information is so optimally compressed, if there is just a little bit of noise in the storage device or communication channel, some information is necessarily lost.

In this module, we consider the opposite setting, where we try to protect our messages from **noise** on a communication channel. The noise may convert the channel input x to some potentially different value y:



The goal is to design encoding and decoding functions that can resist this noise, so that the recovered message $\tilde{W}$ is as close as possible to the original message w. To do so, the encoding x must contain some redundancy, i.e., become slightly longer (whereas for compression we wanted x to become shorter than w).

In this module, we will formally define (communication) channels, and see several examples of codes that can protect against noise (**error-correcting codes**). We

will explore which properties an error-correcting code should satisfy in order to decode to the correct message with high probability.

In the second part of the module, we will explore an even harder requirement on error-correcting codes: what if we want to decode to the correct message with *certainty*? This is known as the **zero-error** setting. It really depends on the structure of the noise in the channel whether that is even possible. We will learn tools to study the noise structure of a channel.

As an additional resource, you may find these student-recorded lectures or these screen recordings useful. However, please remember that the course content is defined by the material available on Canvas, so you may not get all the content by only watching the lecture recordings.

Definition: Discrete Channel

We now make the notion of 'channel' more precise.

Definition: Discrete channel

A (discrete) channel is a tuple $(\mathcal{X}, P_{Y|X}, \mathcal{Y})$ such that $\mathcal{X}$ and $\mathcal{Y}$ are finite sets, and for any $x \in \mathcal{X}$, the function $P_{Y|X=x} : \mathcal{Y} \to [0,1]$ is a probability distribution. $\mathcal{X}$ represents the set of possible inputs, $\mathcal{Y}$ the set of possible outputs, and $P_{Y|X}(y|x)$ is the probability of receiving output y given input x.

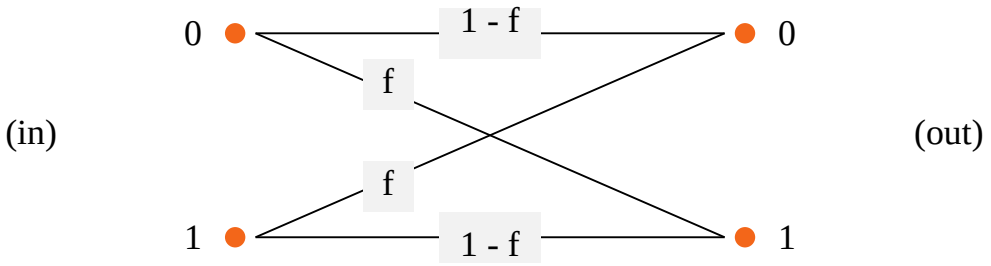
Given a channel $(\mathcal{X}, P_{Y|X}, \mathcal{Y})$, fixing a distribution P_X for the set $\mathcal{X}$, immediately determines a joint distribution P_{XY} and therefore also a distribution P_Y for $\mathcal{Y}$ by marginalization.

Example: Binary symmetric channel (BSC)

Define the binary symmetric channel with parameter $f \in [0, 1/2]$ by $\mathcal{X} = \mathcal{Y} = \{0, 1\}$ and

$$\begin{aligned} P_{Y|X}(0|0) &= P_{Y|X}(1|1) = 1 - f, \\ P_{Y|X}(0|1) &= P_{Y|X}(1|0) = f. \end{aligned}$$

With probability f, the input is flipped, and with probability $1 - f$, it remains unaffected. This channel can be represented visually as:



We are interested in **memoryless** channels where the probability distribution of the output depends only on the current input. If the channel is used repeatedly, the channel distribution does not change depending on previous inputs and outputs. If we use a discrete memoryless channel n times, this can be regarded as the channel $(\mathcal{X}^n, P_{Y^n|X^n}, \mathcal{Y}^n)$ where

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

created: 2019-10-21

Types of Discrete Channels

Definition: Deterministic channel

A channel is deterministic if $H(Y|X) = 0$. In other words,

$$\forall x \in \mathcal{X}\ \exists y \in \mathcal{Y} : P_{Y|X}(y|x) = 1.$$

Definition: Lossless channel

A channel is lossless (or **ideal**) if $H(X|Y) = 0$ for all input distributions P_X. In other words,

$$\forall y \in \mathcal{Y}\ \exists! x \in \mathcal{X} : P_{Y|X}(y|x) > 0.$$

(the notation $\exists! x$ means that there exists *exactly* one such x.)

In a deterministic channel, the output is completely determined by the input, whereas in a lossless channel, the input is completely determined by the output. A **noiseless** channel is a channel that is both deterministic and lossless.

created: 2019-10-21

Definitions: Code, Rate, and Error Probability

In order to get as much information through a channel as possible, we can encode messages before sending them through the channel.

Definition: Code

Let $M, n \in \mathbb{N}$. An (M, n)-code for the channel $(\mathcal{X}, P_{Y|X}, \mathcal{Y})$ consists of

- An index set $[M] = \{1, \dots, M\}$ representing the set of possible messages.
- A (possibly probabilistic) encoding function $\mathbf{enc} : [M] \to \mathcal{X}^n$. This encoding function should be injective. n represents the number of channel uses we need to send a single message.
- A deterministic decoding function $\mathbf{dec} : \mathcal{Y}^n \to [M]$. The set of all codewords, $\{\mathbf{enc}(1), \dots, \mathbf{enc}(M)\}$ is called the **codebook**.

An alternative notation for codes is $[n, k]$ **code**, using box brackets instead of round brackets: such a code encodes a k-bit message into n bits. In the notation of the above definition, an $[n, k]$ code would be an $(2^k, n)$ code.

The number of bits of information that are transmitted per channel use is captured by the following notion:

Definition: Rate

The rate of an (M, n)-code is defined as

$$R := \frac{\log M}{n}.$$

Given a code for a specific channel, we can study the probability that an error occurs while transmitting a message.

Definition: Probability of error

Given an (M, n) code for a channel $(\mathcal{X}, P_{Y|X}, \mathcal{Y})$, the probability of error λ_m is the probability that the decoded output is not equal to the input message m. More formally,

$$\lambda_m^{(n)} := P[\mathbf{dec}(Y^n) \neq m \mid X^n = \mathbf{enc}(m)].$$

Given this quantity, the **maximal probability of error** is defined as

$$\lambda^{(n)} := \max_{m \in [M]} \lambda_m^{(n)}.$$

Similarly, the **average probability of error** is defined as

$$p_e^{(n)} := \frac{1}{M} \sum_{m=1}^{M} \lambda_m^{(n)}.$$

The Repetition Code

A simple and intuitive code is the n-bit **repetition code** R_n: a single message bit is encoded by simply repeating the bit n times. Decoding is done by majority vote, that is, $\mathbf{dec}(y) = MAJ(y_1, \ldots, y_n)$, which is 1 if and only if (strictly) more than half of the bits in y are 1s. In order to avoid ties in the decoding, repetition codes usually require that n is odd.

Example: 3-bit repetition code

Consider the 3-bit repetition code R_3. It is a (2,3) code with codebook $\{000, 111\}$. The rate of R_3 is $1/3$. The probability of error for the message $w = 0$, sent over a BSC with bit flip propability f, is

$$\begin{aligned}\lambda_0 &= P[\mathbf{dec}(Y^n) \neq 0 \mid X^n = 000] \\ &= P[Y^n = 011 \cup Y^n = 101 \cup Y^n = 110 \cup Y^n = 111 \mid X^n = 000] \\ &= 3f^2(1-f) + f^3.\end{aligned}$$

A similar calculation shows that $\lambda_1 = \lambda_0$. Hence, the maximal and average probability of error are equal to λ_0 as well. As a concrete example, if $f = 0.1$, the 3-bit repetition code has an error probability of approximately 0.03. Hence, the 3-bit repetition code provides an error probability that is about three times lower (3% instead of 10%), at the expense of a rate that is a factor 3 worse than simply sending the messages through the channel without encoding.

In general, the n-bit repetition code is a $(2, n)$ code with the relatively low rate of $1/n$ and an average/maximal probability of error of

$$\sum_{k=(n+1)/2}^{n} \binom{n}{k} f^k (1-f)^{n-k}$$

when used on a binary symmetric channel with bit flip probability f.

The [7,4] Hamming Code

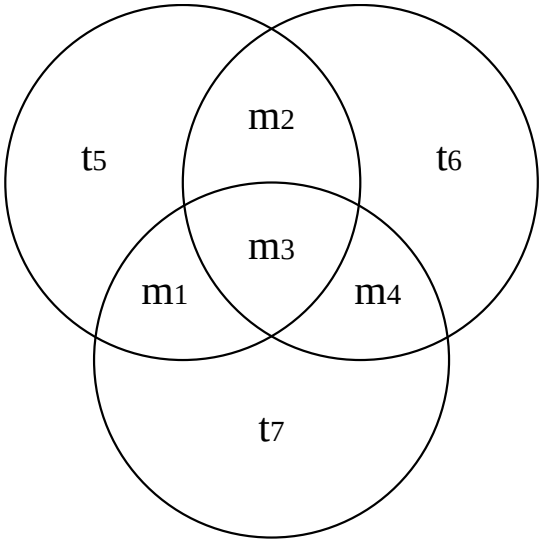
A more sophisticated error-correcting code is the $[7, 4]$ **Hamming code**. It is a $(2^4, 7)$ code, meaning that it encodes a 4-bit message (there are 2^4 such messages) into 7 bits. The encoding function is defined as

$$\texttt{enc}(m_1 m_2 m_3 m_4) = m_1 m_2 m_3 m_4 t_5 t_6 t_7,$$

where the **parity bits**

$$\begin{aligned} t_5 &= m_1 \oplus m_2 \oplus m_3, \\ t_6 &= m_2 \oplus m_3 \oplus m_4, \\ t_7 &= m_1 \oplus m_3 \oplus m_4 \end{aligned}$$

are appended at the right. Note that the choice of these parity bits differs throughout the literature. Decoding is done by making sure that all parity bits check out, and if not, making the smallest possible number of bit flips such that they do. It is important to note that the parity bits themselves may have been flipped during the transmission through the channel. A visual way to perform this parity check is by using the following diagram:



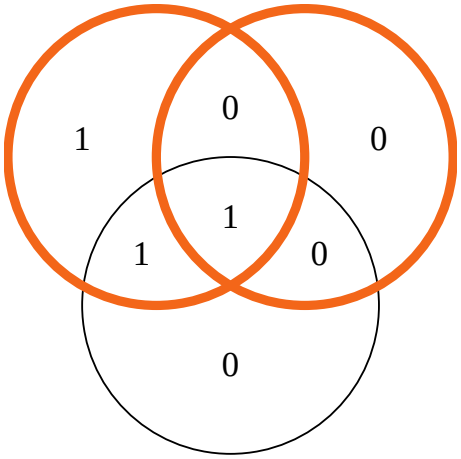
For all of the three circles, the parity bit should equal the parity of the three message bits in that circle. Equivalently, the parity of all bits in a circle should be even.

Example

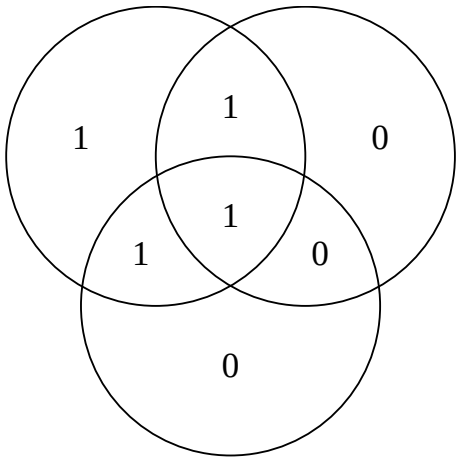
Decode the string 1010100.

Show solution

First, we fill in the bits into the diagram:



We see that in two of the circles, the parity bit is incorrect. This is called the **error syndrome** (you will learn a more formal definition of the error syndrome soon). By flipping only the bit m_2 (which is in the intersection of the top two circles, but not the bottom one), we can fix all the parity bits:



Hence, $\texttt{dec}(1010100) = 1110$.

The $[7, 4]$ Hamming code can correctly decode if the codeword is corrupted in (at most) one place.

Minimal Distance Between Codewords

The following is a distance measure between two strings (or codewords).

Definition: Hamming distance

The Hamming distance between two n-bit strings x and y is defined as

$$d(x,y) := \sum_{i=1}^{n} |x_i - y_i|.$$

One can equivalently define $d(x,y) = |x-y|$ where $|z|$ denotes the **Hamming weight** of a binary string: the number of ones in that string.

The number of bit flips a code can correct depends on the minimal (Hamming) distance between the words in the codebook:

Definition: Minimal distance

Given a binary code with codebook C, the minimal distance of that code is defined as

$$d_{\min} := \min_{\substack{x,y \in C \\ x \neq y}} d(x,y).$$

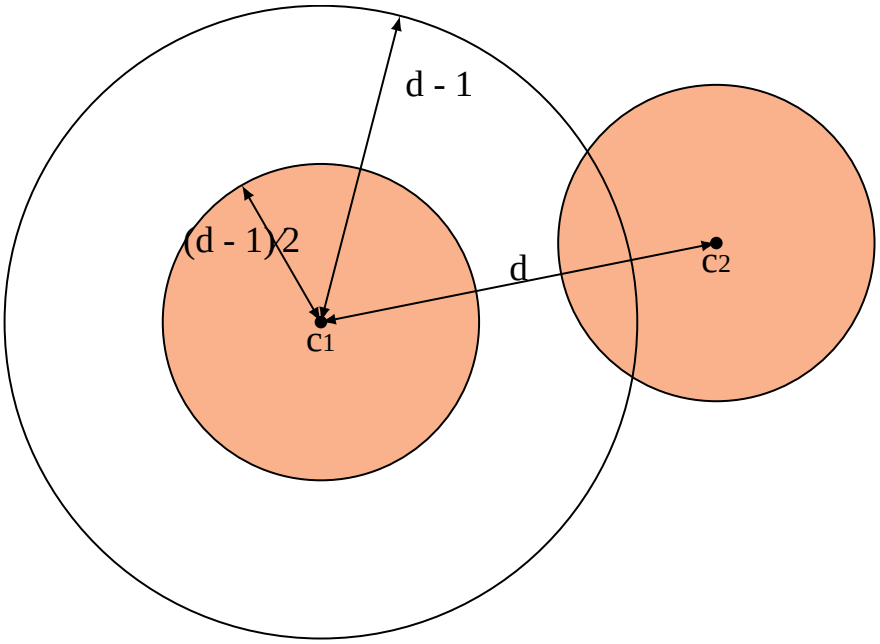
By checking all pairs of codewords of the $[7,4]$ Hamming code, one can verify that its minimal distance is 3 (for this reason, it is often called a $[7,4,3]$ code). Hence, if two bits in a codeword are flipped, it will be closer to some other codeword in terms of number of bit flips. By flipping a single bit, the channel output is (incorrectly) decoded into the message that corresponds to that other codeword.

In general, a code that encodes k input symbols into n output symbols (i.e., that is a $(2^k, n)$ code) and has distance d is often called a $[n,k,d]$ **code**. If the distance is not made specific, it can also be written as an $[n,k]$ **code** (see, for example, the $[7,4]$ Hamming code).

created: 2019-10-21

Error Detection/Correction and the Hamming Bound

In general, a code with minimal distance d can ***detect*** up to $d-1$ bit flip errors: in $d-1$ bit flip 'steps', those bit flips can never result in another valid codeword. The code can accurately ***correct*** up to $\frac{d-1}{2}$ bit flip errors. Look at the diagram below to understand why: if two codewords c_1 and c_2 are guaranteed to be at least d bit flips apart, then the set of strings that result from $\frac{d-1}{2}$ bit flips on c_1 (the orange circle on the left) never overlaps with the set of strings that result from $\frac{d-1}{2}$ bit flips on c_2 (the orange circle on the right).



Because every codeword is guaranteed to have this neighborhood around it that it does not share with any other codeword, we can upper bound the total number of codewords in terms of the distance.

Proposition: Hamming bound

If C is a *binary* code of block length n and minimum distance 3, then $|C| \leq \frac{2^n}{n+1}$.

Proof

For each $c \in C$, define the neighborhood of c to be $N(c) := \{y \in \{0,1\}^n \mid d(x,y) \leq 1\}$. Every such neighborhood contains exactly $n+1$ elements. Since $d = 3$, $N(c) \cap N(c') = \emptyset$ whenever $c \neq c'$. Hence,

$$2^n \geq |\bigcup_{c \in C} N(c)| = \sum_{c \in C} |N(c)| = |C| \cdot (n+1),$$

and the result follows.

The $[7, 4]$ Hamming code is optimal in the sense that it achieves this Hamming bound: it is a code with block length 7 and minimal distance 3, so an upper bound to the codebook size is $\frac{2^7}{7+1} = 2^4$. The Hamming code achieves exactly this codebook size.

Definition: Linear Code

In this section, we study a class of error-correcting codes called linear codes. This type of code has a nice structure and can be encoded/decoded efficiently.

Definition: Linear code

A code C is linear if any linear combination of codewords is also a codeword.

For the definition of linearity to make sense, addition and multiplication by constants needs to be defined on $\mathcal{X}^n$ (formally, $\mathcal{X}^n$ needs to be a vector space). Then C is linear if it is a linear subspace of $\mathcal{X}^n$. In the following, we will assume that $\mathcal{X} = \{0, 1\}$: in that case, we are talking about **binary codes** and addition is simply bitwise addition modulo 2 (which is the exclusive OR function). Note that for binary codes, addition and subtraction are the same operation, as $-1 = +1$ modulo 2.

created: 2019-10-21

Encoding: Generator Matrix

The encoding procedure of a linear code is nicely captured by the generator matrix:

Definition: Generator matrix

Given a $[n, k, d]$ linear code, the generator matrix G^T is an $n \times k$ matrix such that the columns $c_1, \ldots, c_k$ form a basis for C. The generator matrix can always be transformed into **systematic form**:

$$G^T = \begin{pmatrix} \mathbb{I}_k \\ \hline P \end{pmatrix},$$

where P is some $(n-k) \times k$ matrix representing the parity bits of the code.

The generator matrix is used in the encoding function as follows:

$$\mathbf{enc}(m) = G^T \cdot m.$$

The codebook C is the set $\{G^T \cdot m \mid m \in \mathcal{X}^k\}$.

The reason for the transposition in G^T is that historically, coding theorists prefer to use row vectors and matrix multiplication from the right instead of column vectors and multiplication from the left, which is more standard in other areas. Notice that for row vectors $c = m \cdot G$, we equivalently have column vectors $c^T = (m \cdot G)^T = G^T \cdot m^T$.

Example: Generator matrix of the $[7, 4]$ Hamming code

The following 7×4 matrix generates the $[7, 4]$ Hamming code:

$$G^T = \left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{array}\right).$$

The generator matrix is given in systematic form. Encode the message 1010.

created: 2019-10-21

Show solution

To encode the message 1010, we compute

$$G^T \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}.$$

Note that due to the systematic form, the first k bits of the codeword is the actual message, whereas the rest are the parity-check bits.

Decoding: Parity-Check Matrix and Error Syndromes

The decoding operation can be described by defining the parity-check matrix:

Definition: Parity-check matrix

Let C be a code with generator matrix

$$G^T = \begin{pmatrix} \mathbb{I}_k \\ \hline P \end{pmatrix}.$$

Then the parity check matrix for C is given by

$$H := (-P \mid \mathbb{I}_{n-k}).$$

Note that if $\mathcal{X}^n = \{0,1\}^n$, then P is a matrix with binary entries, and hence $-P = P$.

The parity-check matrix shows whether or not an error has occurred, and if it did, what kind of error occurred. The information about the error is contained in the syndrome, which you get by applying the parity check to the received codeword.

Definition: Syndrome

Let C be a code with parity-check matrix H. The syndrome of a received codeword $c \in \mathcal{Y}^n$ is Hc.

For all $c \in C$, the syndrome is the all-zero vector, since $c = G^T m$ for some m, and

$$HG^T m = (-P \mid \mathbb{I}_{n-k}) \begin{pmatrix} \mathbb{I}_k \\ \hline P \end{pmatrix} m = 0^{n-k},$$

where we make use of block matrix multiplication. The syndrome gives a lot of valuable information about where an error occurred. Suppose a codeword c is sent over a channel, and a single bit of c is flipped, at the ith position. The output of the channel is thus $c' = c + e_i$ (where e_i is the ith unit vector, which is 1 at position i and 0 elsewhere). The syndrome for this received output is

$$Hc' = H(c + e_i) = Hc + He_i = He_i,$$

which is the ith column of H. So by comparing the syndrome to the parity-check matrix H, we can find out which bit was most likely flipped.

Example: Parity-check matrix of the $[7,4]$ Hamming code

The following 3×7 matrix is the parity-check matrix for the $[7,4]$ Hamming code:

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

Suppose the codeword 1010100 is received. What is the syndrome? Which errors occurred, if any? And what is the correct decoding?

Show solution

The syndrome is 110: the result of applying H to the received codeword. This syndrome matches He_2 (the second column of H), and so it is most likely the second codeword bit that was flipped. The decoding is therefore 1110 (corrected from 1010).

Notice that all columns of H are different, and that there are $2^3 = 8$ possible syndromes we might observe. This aligns well with the fact that there are three parity bits that can all either be correct or incorrect.

Minimal Distance of Linear Codes

Apart from the trivial way to determine the minimal distance of a code (which is listing the entire codebook and comparing all the codeword pairs), there is a much faster way if the code is linear. It turns out that it already suffices to consider just the Hamming weights of the (nonzero) codewords:

Proposition

For a linear code C, the minimal distance is equal to the minimal weight of the nonzero codewords.

Proof

The following derivation proves the claim:

$$d_{\min} = \min_{\substack{x,y \in C \\ x \neq y}} d(x,y) = \min_{\substack{x,y \in C \\ x \neq y}} \sum_{i=1}^{n} |x_i - y_i| = \min_{\substack{x,y \in C \\ x \neq y}} d(x-y,0) = \min_{\substack{z \in C \\ z \neq 0}} d(z,0) = \min_{\substack{z \in C \\ z \neq 0}} |z| \,,$$

where |z| denotes the Hamming weight of a string z.

An equivalent way to determine the minimal distance of a linear code is possible if the parity check matrix is known.

Proposition

For a linear code C with parity check matrix H, the minimal distance $d_{\min}$ equals the minimum number of columns of H that are linearly dependent.

Proof

Left as an exercise.

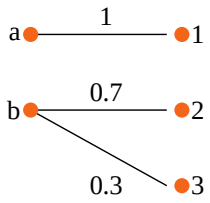
created: 2019-10-21

Introduction: Zero-Error Channel Coding

We consider the problem of using a noisy channel to transmit a message perfectly, i.e., with maximal probability of error equal to zero. For some channels, for example the non-trivial binary symmetric channel with $f \notin \{0, 1\}$, it is not possible to send multiple different messages over the channel in this way. For other channels, an interesting question is: how many messages (or how much information) can be sent over this channel in an error-free way?

Example

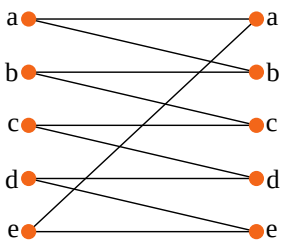
Consider the following channel:



We can send two messages, m_1 and m_2, over the channel by defining $\mathbf{enc}(m_1) = a$ and $\mathbf{enc}(m_2) = b$. The decoding is defined as $\mathbf{dec}(1) = m_1$, and $\mathbf{dec}(2) = \mathbf{dec}(3) = m_2$.

Example: Noisy typewriter

The noisy typewriter channel sends the letters a through e, but with some nonzero probability, it sends the adjacent letter instead. It is defined as follows:



How many messages can you send error-free over this channel?

Show solution

There is a way to send two messages m_1 and m_2 error-free over this channel by defining $\mathbf{enc}(m_1) = a$ and $\mathbf{enc}(m_2) = c$. The decoding function is defined as

$\texttt{dec}(a) = \texttt{dec}(b) = m_1$, and $\texttt{dec}(c) = \texttt{dec}(d) = m_2$ (note that the definition of $\texttt{dec}(e)$ is irrelevant, as this output symbol will never be observed).

Is there a way to encode three different messages in an error-free way? Upon inspection, we see that any encoding function $\texttt{enc}(\cdot)$ on three messages will map at least two messages to channel inputs that are **confusable** (i.e., are possibly mapped to the same channel output).

In general, it is not easy to tell directly from the channel how many messages can be perfectly transmitted. We will invoke some graph theory to help us with the analysis.

Definition: Independence Number

Definition: Undirected Graph

A undirected simple graph G consists of a set $V(G)$ of **vertices** and a set $E(G)$ of **edges**. The edges are unordered pairs of vertices: each edge connects two different vertices of the graph.

Definition: Independence number

The independence number $\alpha(G)$ of a graph G is the size of the largest **independent set** of G, where an independent set of G is a set $S \subseteq V(G)$ such that

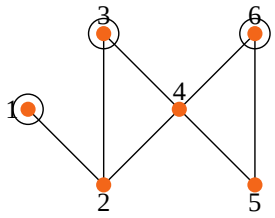
$$\forall x, x' \in S : (x, x') \notin E(G).$$

That is, an independent set S in G is a set of vertices such that there is no edge between any of the vertices.

Finding the independence number of a graph is an NP-hard problem, meaning there is no known efficient method for finding the independence number of an arbitrary graph.

Example

Consider the following graph with $V(G) = \{1, 2, 3, 4, 5, 6\}$ and $E(G) = \{\{1, 2\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}, \{4, 6\}, \{5, 6\}\}$:



A maximal independent set $\{1, 3, 6\}$ is marked in the graph. As there is no independent set of size 4 (can you prove that?), the independence number of this graph is 3.

Definition: Confusability Graph

Definition: Confusability graph

Let $(\mathcal{X}, P_{Y|X}, \mathcal{Y})$ be a channel. The confusability graph G for the channel consists of the set of input symbols of the channel:

$$V(G) := \mathcal{X},$$

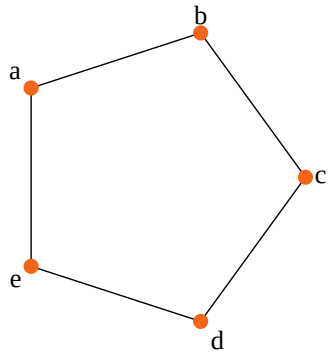
and

$$E(G) := \{\{x, x'\} \subset \mathcal{X} \mid x \neq x' \text{ and } \exists y \in \mathcal{Y} \text{ s.t. } P_{Y|X}(y|x) \cdot P_{Y|X}(y|x') > 0\}$$

is the set of input pairs that are confusable (because they reach a shared output symbol $y \in \mathcal{Y}$).

Example: Confusability graph of the noisy typewriter

Consider again the noisy typewriter channel. The confusability graph for that channel is:



This graph is also known as C_5, the circle of size 5. Its independence number is $\alpha(C_5) = 2$.

In the above example, the independence number of the confusability graph is exactly the number of messages that can be sent over the channel perfectly. This is no coincidence:

Proposition

Given a channel with confusability graph G, the maximal message set $[M]$ that can be communicated perfectly in a single channel use is of size $\alpha(G)$.

Proof

Let $(x, x') \in E(G)$, i.e., x and x' are confusable. They cannot both be used to send different messages, for suppose there are messages $m \neq m'$ such that $\mathbf{enc}(m) = x$ and $\mathbf{enc}(m') = x'$, then by definition of the confusability graph, there is a $y \in \mathcal{Y}$ such that x and x' are both mapped to y with nonzero probability. In order to correctly decode in all cases, it must therefore be that $\mathbf{dec}(y) = m$ and $\mathbf{dec}(y) = m'$, contradicting the assumption that $m \neq m'$. Therefore, the number of messages that can be sent over the channel cannot exceed the independence number $\alpha(G)$.

For the other direction, it is easy to find an encoding and decoding function for $\alpha(G)$ different messages. Let $\{x_1, x_2, \ldots, x_{\alpha(G)}\}$ be a largest independent set of G. Define $\mathbf{enc}(m_i) = x_i$ for all $i \in [\alpha(G)]$. Then for all $y \in \mathcal{Y}$, by definition of the confusability graph and the independent set, there is exactly one i such that $P_{Y|X}(y|x_i) > 0$. Define $\mathbf{dec}(y) = m_i$. This code can send $\alpha(G)$ different messages over the channel without error.

Definition: Strong Graph Product

Definition: Strong graph product

Let G, H be two graphs. We define the strong graph product $G \boxtimes H$ as follows. The set of vertices is

$$V(G \boxtimes H) := V(G) \times V(H).$$

The set of edges is

$$E(G \boxtimes H) := \{\{(x,y),(x',y')\} \mid (x,y) \neq (x',y') \text{ and } (x = x' \text{ or } \{x,x'\} \in E(G)) \text{ and } (y = y' \text{ or } \{y,y'\} \in E($$

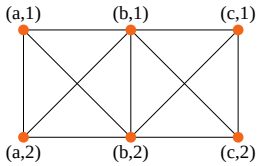
i.e., there is an edge between (x,y) and (x',y') if and only if the vertices of G are confusable (or equal) *and* the vertices of H are confusable (or equal).

Example

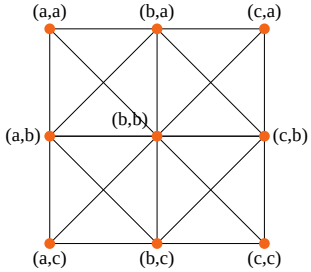
Consider the graph G:

and the graph H:

The strong graph product of $G \boxtimes H$ is



The independence number of this graph is 2. The strong product $G \boxtimes G$ of G with itself is



The independence number of this graph is 4. As the graphs get bigger, the independence number is increasingly hard to compute.

For our application, we will often be interested in the strong graph product of a graph G with itself, possibly many times. Therefore it is useful to work out the definition of $G^{\boxtimes n}$, based on the above definition:

$$V(G^{\boxtimes n}) = V(G) \times \cdots \times V(G)$$
$$E(G^{\boxtimes n}) = \{\{(x_1,\ldots,x_n),(v_1,\ldots,v_n)\} \mid (x_1,\ldots,x_n) \neq (v_1,\ldots,v_n) \text{ and } \forall i : x_i = v_i \vee \{x_i,v_i\} \in E(G)\}$$

Multiple Channel Uses

Earlier in this module we have found that the independence number of the confusability graph is a conclusive upper bound for the number of messages that can be sent over a channel *in a single use*. But it turns out that if the same channel is allowed to be used more than once, it is sometimes possible to achieve a higher rate, still error-free!

Example: Two uses of the noisy typewriter

Let us reconsider the noisy typewriter, but this time we are allowed to use the channel twice. We can of course use the following encoding function to encode $2 \times 2 = 4$ different messages:

$$\begin{aligned}\texttt{enc}(m_1) &= \texttt{aa}\\ \texttt{enc}(m_2) &= \texttt{ac}\\ \texttt{enc}(m_3) &= \texttt{ca}\\ \texttt{enc}(m_4) &= \texttt{cc}\end{aligned}$$

This encoding function is defined by simply concatenating the encoding function for the single-use example. It has a rate of $\log(4)/2 = 1$, just like the single-use noisy typewriter channel code. There is, however, a code that is able to encode and correctly decode *five* different messages! It uses codewords $\texttt{aa}, \texttt{bc}, \texttt{ce}, \texttt{db}, \texttt{ed}$ as codewords for messages m_1 through m_5. To see that the decoding is error-free, observe that the channel maps the codewords to the following sets of possible outputs:

$$\begin{aligned}\texttt{aa} &\mapsto \{\texttt{aa}, \texttt{ab}, \texttt{ba}, \texttt{bb}\}\\ \texttt{bc} &\mapsto \{\texttt{bc}, \texttt{bd}, \texttt{cc}, \texttt{cd}\}\\ \texttt{ce} &\mapsto \{\texttt{ce}, \texttt{ca}, \texttt{de}, \texttt{da}\}\\ \texttt{db} &\mapsto \{\texttt{db}, \texttt{dc}, \texttt{eb}, \texttt{ec}\}\\ \texttt{ed} &\mapsto \{\texttt{ed}, \texttt{ee}, \texttt{ad}, \texttt{ae}\}\end{aligned}$$

All of these sets are non-overlapping, so none of the inputs are confusable. The rate of this code is $\log(5)/2 \approx 1.161$, which is strictly better than the single-use channel!

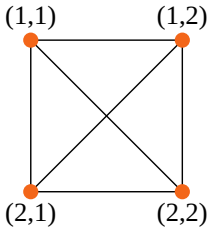
The above example shows that there are optimal codes that make better use of the channel if the channel is used more than once. We can determine whether this is the case by looking at the strong graph product of the channel's confusability graph with itself. An edge in the strong graph product reflects that the two input

pairs are confusable: there exists an output pair to which they are both mapped with nonzero probability.

Example: Binary symmetric channel

The confusability graph of the non-trivial binary symmetric channel (BSC) is C_2:

This graph is also known as K_2, the **complete graph** of size 2. $K_2 \boxtimes K_2$ is the complete graph on 4 vertices K_4:



In this case, using the channel twice does not help: the independence number of both graphs is 1, and the rate of any error-free code is therefore zero.

Shannon Capacity of a Graph

In the previous section it became apparent that for some channels, two or more channel uses can provide a better rate than a single use of those channels. The maximum rate that can be achieved in this way is captured by the notion of Shannon capacity of the confusability graph of the channel.

Definition: Shannon capacity of a graph

The Shannon capacity of a graph is

$$c(G) := \sup_{n\in\mathbb{N}} \frac{\log \alpha(G^{\boxtimes n})}{n}.$$

The Shannon capacity represents the *maximum number of bits per channel use* that can be perfectly communicated over a channel with confusability graph G. Note that our definition in terms of message bits differs from the definition more commonly used in the literature (and on wikipedia) about zero-error communication where the capacity is defined without the log as $\sup_{n\in\mathbb{N}} \sqrt[n]{\alpha(G^{\boxtimes n})}$, thus measuring the maximum number of actual messages (and not bits) per channel use that can be perfectly communicated.

Intuitively, allowing more channel uses can only increase the rate. This is reflected by the fact that we can replace the supremum with a limit, as captured by the following lemma.

Proposition

$$c(G) = \lim_{n\to\infty} \frac{\log \alpha(G^{\boxtimes n})}{n}.$$

Proof

First, note that $\alpha(G^{\boxtimes(k+\ell)}) \geq \alpha(G^{\boxtimes k})\alpha(G^{\boxtimes \ell})$ (you showed this inequality in the previous quiz). It then follows that

$$\log \alpha(G^{\boxtimes(k+\ell)}) \geq \log \alpha(G^{\boxtimes k}) + \log \alpha(G^{\boxtimes \ell}),$$

and so the sequence $(\log \alpha(G^{\boxtimes n}))_{n\in\mathbb{N}}$ is superadditive. Then by Fekete's lemma,

$$\lim_{n\to\infty} \frac{\log \alpha(G^{\boxtimes n})}{n}$$

exists and is equal to the supremum.

The exact computational complexity of $c(G)$ is unknown. It is not even known to be a decidable problem. We have seen that $c(C_5)$ is at least $\frac{\log 5}{2}$, a fact that has been known since Shannon showed it in 1956, but how can we decide whether the capacity is actually bigger than that number? This question remained open until Lovasz showed in 1979 that the Shannon capacity of C_5 is exactly $c(C_5) = \frac{\log 5}{2}$. For the relatively small circle of size 7, C_7, the exact Shannon capacity remains unknown. From the fact that $c(C_5) = \frac{\log 5}{2}$, we know that the 5-letter noisy typewriter does not benefit from more than two channel uses. In fact, all graphs for which the Shannon capacity is known attain that capacity with one, two or an infinite number of channel uses. It is not known if there is a deeper reason for this observed pattern.

It is quite remarkable that in this rather simple and natural problem of zero-error channel coding, we quickly reach the limit of our understanding of the underlying combinatorial problems.

Definition: Channel Capacity

We just discovered that for some noisy channels, zero-error communication is very hard, or even impossible. For example, if Alice and Bob have to communicate over a binary symmetric channel (BSC) that has non-zero bit-flip probability, they cannot hope to do any zero-error communication, because the Shannon capacity of the BSC's confusability graph is zero.

We also saw that error-correcting codes can help deal with such inherently noisy channels. Even though the communication error may not become zero, an error-correcting code can increase the probability of receiving the correct message. It does come at a cost, however, because the codewords are longer than the original messages, and so the amount of information that is transmitted *per channel use* does not necessarily increase.

In this final part of the module, we explore the limits of how much information can be sent over a channel if a small error is allowed. Central to our study will be the concept of channel capacity. It reflects the maximum amount of information that could *in principle* be communicated with a single use of a channel. In the next module, we will see how well that theoretical limit can be approached with actual error-correcting codes.

Definition: Channel capacity

The channel capacity C of a discrete, memoryless channel $(\mathcal{X}, P_{Y|X}, \mathcal{Y})$ is given by

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

Remember that using a certain input distribution P_X for a channel $P_{Y|X}$ yields a joint input-output distribution P_{XY} which determines the real quantity $I(X;Y)$ we can optimize over. One can argue that the maximum is attained and therefore the channel capacity is a well-defined quantity.

Important: the channel capacity is often called the Shannon capacity (of a channel). You should not confuse it with the Shannon Capacity of a Graph.

created: 2019-10-21

Generally, the Shannon capacity of a channel is not equal to the Shannon capacity of its confusability graph.

Example: Capacity of a BSC

What is the capacity (in terms of f) of a binary symmetric channel with parameter $f \in [0, 1/2]$?

Show hint

Rewrite $I(X;Y)$ as $H(Y) - H(Y|X)$ and note that you can compute $H(Y|X)$ without fixing P_X. Then think about how to maximize $H(Y)$.

Show solution

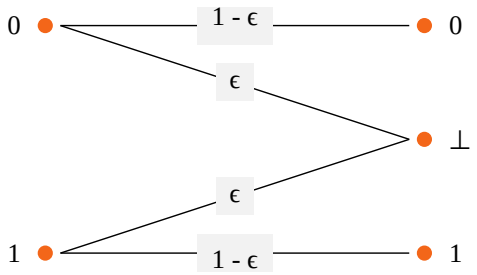
The channel capacity is

$$\begin{aligned}\max_{P_X} I(X;Y) &= \max_{P_X} (H(Y) - H(Y|X)) \\ &= \max_{P_X} \left(H(Y) - \sum_{x \in \mathcal{X}} P_X(x) \cdot H(Y|X=x) \right) \\ &= \max_{P_X} \left(H(Y) - \sum_{x \in \mathcal{X}} P_X(X) \cdot h(f) \right) \\ &= \max_{P_X} (H(Y) - h(f)) \\ &= 1 - h(f).\end{aligned}$$

The last step follows because $H(Y)$ is maximized if Y is uniform, which is achievable by choosing X to be uniform.

Example: Capacity of a BEC

Consider the binary erasure channel (BEC) with $\mathcal{X} = \{0, 1\}$ and $\mathcal{Y} = \{0, 1, \perp\}$, where $\perp$ is the **erasure symbol**, and $\epsilon \in [0, 1]$ is the **erasure probability**:



What is the channel capacity of the BEC, as a function of ϵ?

Show hint

Contrary to the previous example, break $I(X;Y)$ up as $H(X) - H(X|Y)$, using symmetry of the mutual information. Consider the three possible outputs

separately: how much uncertainty is left if you receive output 0? What about output 1? And output $\perp$?

Show solution

Write p for $P_X(0)$.

$$\begin{aligned}\max_{P_X} I(X;Y) &= \max_{P_X} (H(X) - H(X|Y)) \\ &= \max_{p} \left(h(p) - \sum_{y \in \mathcal{Y}} P_Y(y) \cdot H(X|Y=y) \right) \\ &= \max_{p} (h(p) - P_Y(\perp) \cdot h(p)) \\ &= \max_{p} (h(p)(1-\epsilon)) \\ &= 1 - \epsilon \,.\end{aligned}$$

Again, the last step follows because $H(X) = h(p)$ is maximized if X is uniform, hence $p = \frac{1}{2}$.

If a channel is memoryless, then using it more than once does not increase the capacity *per transmission*. Note that this is different from the zero-error setting, where multiple channel uses can in fact increase the efficiency of getting information across! This is formally captured in the following lemma, which we state without proof:

Lemma: Multiple Channel Uses

Let $X_1, \ldots, X_n =: X^n$ be n random variables. Let Y^n be the result of passing X^n through a discrete memoryless channel of capacity C. Then for any joint distribution P_{X^n},

$$I(X^n, Y^n) \leq n \cdot C.$$

Introduction to Module 07

Previous modules:

- 01: Preliminaries (probability theory)
- 02: Building blocks (entropy, conditional entropy, mututal information, and relative entropy)
- 03/04: *compressing* information: encoding sources symbol-by-symbol (Huffman codes, arithmetic codes, Shannon's source coding theorem about the average codeword length of optimal codes), and encoding sources in blocks (typical sets).
- 04: *hiding* information: perfectly secure encryption (one-time pad).
- 05: generalization to stochastic processes (entropy rate)
- 06: *protecting* information: error-correcting codes and zero-error coding

In this final module, we will try to find the optimal balance between compressing information on the one hand (so that we may send messages as efficiently as possible), and protecting it from noise on the other hand.

The channel capacity plays an important role in this balance. As stated, the channel capacity reflects the maximum amount of information that could *in principle* be sent over a noisy channel per use of that channel. The question remains whether this capacity is actually **achievable**.

We will prove **Shannon's noisy-channel coding theorem**, which states that any rate R that is strictly below the capacity C is achievable, and conversely, that any rate strictly above is not achievable. In the proof of Shannon's noisy-channel coding theory, the concept of typicality (and a variant of the AEP) will play an important role.

We end the module with a final theorem, the **source-channel separation theorem**, which states that a good method (as in: efficient and protecting against errors) to encode a source to send it over a channel is to *first* compress it down, and *then* apply an error-correcting code that is appropriate for the channel.

As an additional resource, you may find these screen recordings useful. However, please remember that the course content is defined by the material available on

Canvas, so you may not get all the content by only watching the lecture recordings.

Definition: Achievable Rate

As stated, the channel capacity reflects the maximum amount of information that could *in principle* be sent over a noisy channel per use of that channel. The question remains whether this capacity is **achievable** by an actual code in the following sense:

Definition: Achievable rate

For a given channel, a rate R is achievable if there exists a sequence of $(2^{n \cdot R}, n)$ codes (for $n = 1, 2, 3, \ldots$) such that $\lambda^{(n)} \xrightarrow{n \to \infty} 0$. Here, $\lambda^{(n)}$ is the maximum error probability as defined in the previous module.

Note that any $(2^{n \cdot R}, n)$ code has rate R, since $\frac{1}{n} \cdot \log 2^{n \cdot R} = R$. Thus, a rate R is achievable if there exists a sequence of rate R codes such that increasing the number of channel uses n (often called the block size) reduces the maximum error asymptotically to zero.

This final module will be concerned with proving **Shannon's noisy-channel coding theorem**, which states that any rate R that is strictly below the capacity C is achievable, and conversely, that any rate strictly above is not achievable.

created: 2019-10-21

Fano's Inequality

Suppose you see Y, the output of some noisy channel, and you want to guess what the input to the channel must have been. Let your guess $\hat{X}$ be some function of your observation of Y, that is, $\hat{X} = g(Y)$. Note that $X \to Y \to \hat{X}$ forms a Markov chain.

Fano's inequality relates the probability that your guess is wrong $P[\hat{X} \neq X])$ to $H(X|Y)$: the uncertainty you have about the channel's input X when you are only given the output Y.

Theorem: Fano's Inequality

Let P_{XY} an arbitrary joint distribution of random variables X and Y, and let $\hat{X} = g(Y)$ for some function g. Furthermore, define $p_e := P[\hat{X} \neq X]$ to be the probability of error. Then

$$H(X|Y) \leq p_e \cdot \log(|\mathcal{X}| - 1) + h(p_e).$$

Since we know that $0 \leq p_e \leq 1$, and thus $h(p_e) \leq 1$ we may rewrite Fano's inequality as

$$p_e \geq \frac{H(X|Y) - 1)}{\log(|\mathcal{X}| - 1)}.$$

Proof

Define the random variable E to be 0 whenever $\hat{X} = X$, and 1 otherwise. (In other words, E indicates whether an error has occurred in guessing the input.)

Observe the following relations between E, X, and $\hat{X}$:

1. $H(E|X\hat{X}) = 0$ (since E is a function of X and $\hat{X}$).
2. $H(E|\hat{X}) \leq H(E) = h(p_e)$ (by general properties of conditional entropy).
3. $H(X|\hat{X}, E = 0) = 0$ (if you know that the guess was correct, you can infer the original input from the guess).
4. $H(X|\hat{X}, E = 1) \leq \log(|\mathcal{X}| - 1)$ (if you know that the guess was incorrect, at least you know that the correct input was one of the $|\mathcal{X}| - 1$ other options).

These observations allow us to derive the inequality:

$$\begin{aligned}
H(X|Y) &\leq H(X|\hat{X}) && \text{(by the data-processing ine} \\
&= H(E|\hat{X}) + H(X|E\hat{X}) && \text{(by entropy diagrams and observat} \\
&\leq h(p_e) + H(X|E\hat{X}) && \text{(by observat} \\
&= h(p_e) + P_E(0) \cdot H(X|\hat{X}, E = 0) + P_E(1) \cdot H(X|\hat{X}, E = 1) \\
&= h(p_e) + 0 + P_E(1) \cdot H(X|\hat{X}, E = 1) && \text{(by observat} \\
&\leq h(p_e) + p_e \cdot \log(|\mathcal{X}| - 1) && \text{(by observati}
\end{aligned}$$

Definition: Joint Typicality

Very roughly, the forward direction of Shannon's noisy-channel coding theorem follows from the asymptotic equipartition property (AEP). Increasing n (the size of the input blocks) results in output blocks that are more likely to be typical. But can we be sure that no two typical input sequences result in the same typical output sequence? Such a collision would make it impossible to decode correctly. To show that this does not form a problem, we need stronger mathematical tools.

Before you start on this section, it is a good idea to refresh your understanding of typical sets and the asymptotic equipartition property.

Definition: Joint typicality

For a joint distribution P_{XY}, the jointly typical set $A_\epsilon^{(n)}$ is defined as

$$A_\epsilon^{(n)} := \{(x^n, y^n) \in \mathcal{X}^n \times \mathcal{Y}^n | \quad \left|-\frac{1}{n}\log P_{X^n}(x^n) - H(X)\right| < \epsilon, \\ \left|-\frac{1}{n}\log P_{Y^n}(y^n) - H(Y)\right| < \epsilon, \\ \left|-\frac{1}{n}\log P_{X^nY^n}(x^n, y^n) - H(XY)\right| < \epsilon\}.$$

Here, $P_{X^nY^n} := \prod_{i=1}^n P_{XY}(x_i y_i)$.

A jointly typical set is a set where both elements of the pair are typical individually under the marginal distributions, and their combination is typical under the joint distribution. To get some intuition for what a typical set looks like, we give two examples: one where X and Y are very correlated, and one where they are independent.

Example

Consider the almost fully correlated random variables X and Y with the following distribution:

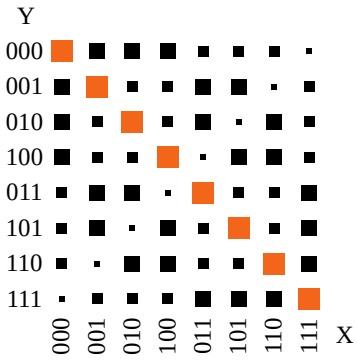
	$X=0$	$X=$
$Y=0$	0.45	0.05

Note that the marginal distributions of X and Y are uniform, and so for all n and all $x^n \in \{0,1\}^n$,

$$-\frac{1}{n}\log P_{X^n}(x^n) - H(X) = -\frac{1}{n}\log\frac{1}{2^n} - \log 2 = 0,$$

and a similar statement holds for Y. Hence, the first and second inequalities in the definition of joint typicality are always satisfied for any $\epsilon > 0$.

What about the third inequality? The table below visualizes the probabilities of every pair for $n = 3$ and $\epsilon = 0.35$: every square represents a pair of elements from $\mathcal{X}^3$ and $\mathcal{Y}^3$. Bigger squares represent higher probabilities. The typical pairs are marked in orange.



As you might expect, pairs (x^3, y^3) with a higher overlap in bits have a higher probability of occurring. The jointly typical set has a very clear structure: only those pairs with $x^3 = y^3$ are included. Increasing ϵ would result in a bigger typical set. For example, for $\epsilon = 0.8$, all pairs that differ in at most one position (e.g., 000, 001) are part of the jointly typical set. Increasing n would have the same effect: for example, for $n = 10$ and $\epsilon = 0.35$, all pairs of sequences that differ at 0, 1, or 2 positions are part of the jointly typical set.

Similarly to what we saw earlier in the course, in this example, for big n and small ϵ, the jointly typical set contains only those pairs of sequences that differ in roughly 10 percent of the positions. The highest-probability pairs are excluded from the typical set. This might match your expectation, since the probability of sampling a non-matching pair of bits ((0,1) or (1,0)) is 10 percent: pairs with that percentage of non-matching bits are 'typical'. These jointly typical pairs together make up almost all of the probability mass, even though individually they are not the highest-probability pairs under P_{XY}.

Example

Consider the following distribution on X and Y:

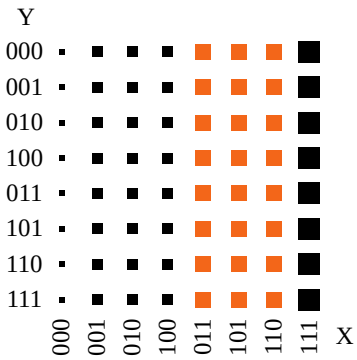
	$X = 0$	$X =$
$Y = 0$	1/8	1/8

Note that X is uniform, and that X and Y are independent. Just like in the previous example, since X is uniform, the first inequality in the definition of joint typicality is always satisfied for any $\epsilon > 0$. Furthermore, since X and Y are independent, the third inequality is satisfied whenever the second inequality is, because

$$\begin{aligned} -\frac{1}{n}\log P_{X^nY^n}(x^n,y^n) - H(XY) &= -\frac{1}{n}\log P_{X^n}(x^n) - \frac{1}{n}\log P_{Y^n}(y^n) - H(X) - H(Y) \\ &= -\frac{1}{n}\log P_{Y^n}(y^n) - H(Y). \end{aligned}$$

Hence, for this distribution, any pair where the second element is typical for Y is also jointly typical.

More concretely, consider $n = 3$ and $\epsilon = 0.35$. For these values, the sequences $011, 101$, and 110 are typical for Y. This results in the following jointly typical set (again, bigger squares represent bigger probabilities, and orange squares indicate elements of the typical set, **typo: the label of the axes label be switched around!**):



Note that $P_Y(111) \approx 0.422$, which is higher than $P_Y(011) \approx 0.141$. However, 111 is not typical for Y while 011 is. 111's probability deviates too far from the 'benchmark' set by $2^{-3H(Y)} \approx 0.185$. Together, the sequences in the typical set have a probability of about 0.422 of occurring. As n increases, this probability will grow, and the total probability of the elements that fall outside of the typical set (such as the all-1 string) will diminish.

Joint Asymptotic Equipartition Property (Joint AEP)

In the examples on the previous page, we have gathered some intuition about jointly typical sets. A (jointly) typical set is usually relatively small compared to the total sample space, but for large n, much of the probability lies on its elements.

We also see the effect of mutual information between X and Y. If X and Y are close to independent, the typical set has a rectangular shape: independently selecting a typical X sample and a typical Y sample will certainly yield a jointly typical element. On the other hand, if X and Y are highly dependent on each other, the typical set has a diagonal structure: there is a significant probability that sampling a typical X and a typical Y independently yields a pair that is not jointly typical. These observations lead us to a joint version of the asymptotic equipartition property, describing some properties of jointly typical sets:

Theorem: Joint Asymptotic Equipartition Property (Joint AEP)

Let X^nY^n be i.i.d. with respect to P_{XY}. Then:

1. $P_{X^nY^n}(A_\epsilon^{(n)}) \xrightarrow{n\to\infty} 1$.
2. $|A_\epsilon^{(n)}| \leq 2^{n(H(XY)+\epsilon)}$.
3. For random variables $\tilde{X}$ and $\tilde{Y}$, if $\tilde{X}^n\tilde{Y}^n$ is i.i.d. with respect to $P_X \cdot P_Y$, then
$$P[(\tilde{X}^n\tilde{Y}^n) \in A_\epsilon^{(n)}] \leq 2^{-n(I(X;Y)-3\epsilon)}.$$
4. For random variables $\tilde{X}$ and $\tilde{Y}$, if $\tilde{X}^n\tilde{Y}^n$ is i.i.d. with respect to $P_X \cdot P_Y$, then for large enough n,
$$P[(\tilde{X}^n\tilde{Y}^n) \in A_\epsilon^{(n)}] \geq (1-\epsilon)2^{-n(I(X;Y)+3\epsilon)}.$$

Proof

We show the statements separately, for an arbitrary X^nY^n that is i.i.d. with respect to P_{XY}.

1. The first statement follows from the weak law of large numbers. Similarly to the proof of the asymptotic equipartition property, we have that

$$\begin{aligned} -\frac{1}{n}\log P_{X^n}(X^n) &= -\frac{1}{n}\sum_{i=1}^{n}\log P_X(X) \\ &\xrightarrow{p} -\mathbb{E}[\log P_X(X)] \\ &= H(X). \end{aligned}$$

Writing out the definition of convergence in probability, we have that for all $\epsilon > 0$, there exists an $n_1 \in \mathbb{N}$ such that for all $n > n_1$,

$$\begin{aligned} P\left[\left|-\frac{1}{n}\log P_{X^n}(X^n) - H(X)\right| \geq \epsilon\right] &\leq P\left[\left|-\frac{1}{n}\log P_{X^n}(X^n) - H(X)\right| \geq \frac{\epsilon}{3}\right] \\ &< \frac{\epsilon}{3}. \end{aligned}$$

In the above, we can divide ϵ by 3 because the second inequality holds for *all* $\epsilon > 0$, so in particular also for $\frac{\epsilon}{3}$. We can derive statements about Y^n and X^nY^n that are similar to this bound above. Doing so, we find $n_2, n_3 \in \mathbb{N}$ for Y^n and X^nY^n, respectively. For all $\epsilon > 0$, we can now choose $n_0 := \max\{n_1, n_2, n_3\}$, and find that for all $n > n_0$,

$$\begin{aligned} P\left[(X^n, Y^n) \in A_\epsilon^{(n)}\right] &= 1 - P\left[(X^n, Y^n) \notin A_\epsilon^{(n)}\right] \\ &\geq 1 - \Bigg(P\left[\left|-\frac{1}{n}\log P_{X^n}(X^n) - H(X)\right| \geq \epsilon\right] + \\ &\qquad P\left[\left|-\frac{1}{n}\log P_{Y^n}(Y^n) - H(Y)\right| \geq \epsilon\right] + \\ &\qquad P\left[\left|-\frac{1}{n}\log P_{X^nY^n}(X^nY^n) - H(XY)\right| \geq \epsilon\right]\Bigg) \\ &\leq 1 - \left(\frac{\epsilon}{3} + \frac{\epsilon}{3} + \frac{\epsilon}{3}\right) \\ &= 1 - \epsilon. \end{aligned}$$

The first inequality is due to the union bound, and the second follows from the inequality in the proof of the previous point, and its analogues. By definition of convergence, the first statement is proven.

2. Observe that by rewriting the last line in the definition of $A_\epsilon^{(n)}$ in the definition of joint typicality, one can derive that for every element $(x^n, y^n) \in A_\epsilon^{(n)}$,

$$P_{X^nY^n}(x^n y^n) > 2^{-n(H(XY)+\epsilon)}.$$

The second statement follows by rearranging the following inequality:

$$\begin{aligned} 1 &= \sum_{(x^n,y^n)\in\mathcal{X}^n\times\mathcal{Y}^n} P_{X^nY^n}(x^n,y^n) \\ &\geq \sum_{(x^n,y^n)\in A_\epsilon^{(n)}} P_{X^nY^n}(x^n,y^n) \\ &\geq |A_\epsilon^{(n)}| \cdot 2^{-n(H(XY)+\epsilon)}. \end{aligned}$$

3. First, we write out the probability of sampling an element in the typical set, again by rearranging the inequalities in the definition of joint typicality.

$$\begin{aligned} P[(\tilde{X}^n\tilde{Y}^n) \in A_\epsilon^{(n)}] &= \sum_{(x^n,y^n)\in A_\epsilon^{(n)}} P_{\tilde{X}^n\tilde{Y}^n}(x^n,y^n) \\ &= \sum_{(x^n,y^n)\in A_\epsilon^{(n)}} P_{X^n}(x^n)\cdot P_{Y^n}(y^n) \\ &\leq |A_\epsilon^{(n)}| \cdot 2^{-n(H(X)-\epsilon)} \cdot 2^{-n(H(Y)-\epsilon)}. \end{aligned}$$

   Combining this with part (2) of this theorem, we get

$$\begin{aligned} P[(\tilde{X}^n\tilde{Y}^n) \in A_\epsilon^{(n)}] &\leq 2^{n(H(XY)+\epsilon)} \cdot 2^{-n(H(X)-\epsilon)} \cdot 2^{-n(H(Y)-\epsilon)} \\ &= 2^{-n(-H(XY)+H(X)+H(Y)-3\epsilon)} \\ &= 2^{-n(I(X;Y)-3\epsilon)}. \end{aligned}$$

   This completes the proof of the third part.

4. By part (1), we have that $P[A_\epsilon^{(n)}] \geq 1-\epsilon$ for large enough n. Thus,

$$\begin{aligned} 1-\epsilon &\leq P[A_\epsilon^{(n)}] \\ &= \sum_{(x^n,y^n)\in A_\epsilon^{(n)}} P_{X^nY^n}(x^n,y^n) \\ &\leq |A_\epsilon^{(n)}| 2^{-n(H(XY)-\epsilon)}. \end{aligned}$$

   Rearranging this inequality and using the same type of derivation as in the proof of (3), it follows that

$$\begin{aligned} P[(\tilde{X}^n\tilde{Y}^n) \in A_\epsilon^{(n)}] &= \sum_{(x^n,y^n)\in A_\epsilon^{(n)}} P_{X^n}(x^n)\cdot P_{Y^n}(y^n) \\ &\geq (1-\epsilon)\cdot 2^{n(H(XY)-\epsilon)} \cdot 2^{-n(H(X)+\epsilon)} \cdot 2^{-n(H(Y)+\epsilon)} \\ &= (1-\epsilon)\cdot 2^{-n(I(X;Y)+3\epsilon)}. \end{aligned}$$

Noisy-Channel Theorem: Forward Direction

With the tools from the previous section, we are ready to prove the forward direction of Shannon's noisy-channel coding theorem, which states that any rate strictly below the channel capacity is achievable:

Theorem: Shannon's noisy-channel coding theorem (forward direction)

For a discrete memoryless channel with capacity C, any rate $R < C$ is achievable. Concretely, for any $\varepsilon > 0$ and any rate $R < C$, for large enough n there exists a $(2^{n \cdot R}, n)$ code with maximal error $\lambda^{(n)} < \varepsilon$.

Proof

Given a channel $(\mathcal{X}, P_{Y|X}, \mathcal{Y})$ with capacity $C = \max P_X I(X;Y)$, let $R < C$ and $\epsilon > 0$. We will first show that for big enough n, a *randomly constructed* code with rate R has a low error probability. We will then argue that a low error probability on average of codes implies the existence of some *specific* code with low error probability.

Fix an input distribution P_X that maximizes $I(X;Y)$. For any n, construct a $(2^{n \cdot R}, n)$-code $\mathcal{C}$ by choosing a codebook at random according to P_X. That is, for every message $w \in [2^{n \cdot R}]$, sample n times from the distribution P_X, creating a codeword $\mathcal{C}(w) = (\mathcal{C}_1(w), \mathcal{C}_2(w), \ldots, \mathcal{C}_n(w))$ by concatenating the n independent samples $\mathcal{C}_i(w) \sim P_X$.

Since the channel is memoryless, if w is sent over the channel using $\mathcal{C}$, the output distribution Y^n is given by:

$$P_{Y^n|X^n}(y^n|\mathcal{C}(w)) = \prod_{i=1}^{n} P_{Y|X}(y_i \mid \mathcal{C}_i(w)).$$

What is the probability that the decoded message $\hat{w}$ is incorrect, i.e., not equal to w? This depends on the decoding method used by the receiver. The optimal decoding procedure is **maximum-likelihood decoding**, where the input message that is most likely with respect to $P_{X|Y}$ is selected as the decoding $\hat{w}$. However, it is hard to analyze the error probability for this decoding method. Instead, we will

assume that the receiver applies **jointly typical decoding**, which has a slightly higher probability of decoding to the wrong message, but still small enough for our analysis. Jointly typical decoding works as follows: upon receiving an output y^n, the receiver looks for a *unique* message $\hat{w}$ such that the pair $(\mathcal{C}(\hat{w}), y^n)$ is jointly typical. If there exists no such message, or if it is not unique, the receiver declares a failure by decoding to $\hat{w} = 0$ (which is always wrong because $w \in [2^{n \cdot R}] = \{1, 2, \ldots, 2^{n \cdot R}\}$).

With this decoding procedure in mind, we analyze the average error probability $P[\texttt{error}]$, where the average is taken over both the randomly constructed code $\mathcal{C}$ and the uniformly randomly selected message w. Defining $\lambda_w(\mathcal{C}) := P[\hat{w} \neq w \mid \mathcal{C}(w) \text{ was sent over the channel}]$ to be the probability that a message w (encoded using $\mathcal{C}$) is decoded incorrectly, we get:

$$\begin{aligned} P[\texttt{error}] &= \sum_{\mathcal{C}} P[\mathcal{C}] \cdot \left(\sum_{w=1}^{2^{n \cdot R}} \frac{1}{2^{n \cdot R}} \cdot \lambda_w(\mathcal{C}) \right) \\ &= \frac{1}{2^{n \cdot R}} \sum_{w=1}^{2^{n \cdot R}} \sum_{\mathcal{C}} P[\mathcal{C}] \cdot \lambda_w(\mathcal{C}). \end{aligned}$$

Since we average over all randomly constructed codes $\mathcal{C}$, and the codewords for all messages are sampled independently, the value $\sum_{\mathcal{C}} P[\mathcal{C}] \cdot \lambda_w(\mathcal{C})$ does not depend on the particular message w. Hence if we set, for example, $w_0 = 1$, then for all $w \in [2^{n \cdot R}]$,

$$\sum_{\mathcal{C}} P[\mathcal{C}] \lambda_w(\mathcal{C}) = \sum_{\mathcal{C}} P[\mathcal{C}] \lambda_{w_0}(\mathcal{C}).$$

This simplifies the calculation of $P[\texttt{error}]$ significantly:

$$\begin{aligned} P[\texttt{error}] &= \frac{1}{2^{n \cdot R}} \sum_{w=1}^{2^{n \cdot R}} \sum_{\mathcal{C}} P[\mathcal{C}] \cdot \lambda_{w_0}(\mathcal{C}) \\ &= \sum_{\mathcal{C}} P[\mathcal{C}] \cdot \lambda_{w_0}(\mathcal{C}). \end{aligned}$$

That is, the average probability of error is the probability (over the selection of the code $\mathcal{C}$, and over the randomness in the channel) that the message w_0 is decoded incorrectly. There are two possible reasons for an error in the decoding:

1. The output of the channel is not jointly typical with $\mathcal{C}(w_0)$. By the first item of the joint AEP, this probability approaches zero as n goes to infinity. Hence, for big enough n, the probability of an error for this reason is smaller than ϵ.

2. There is some $w' \neq w_0$ such that the output of the channel is (also) jointly typical with $\mathcal{C}(w')$. Since $\mathcal{C}$ is a random code (and so $\mathcal{C}(w')$ is independent from the channel output y^n), by the third item of the joint AEP the probability that this occurs is at most

$$\sum_{w' \neq w_0} 2^{-n(I(X;Y)-3\epsilon)} = (2^{n \cdot R} - 1) 2^{-n(I(X;Y)-3\epsilon)}.$$

We can thus bound the average probability of error, using the union bound and the bounds in the above analysis, by

$$\begin{aligned} P[\texttt{error}] &\leq \epsilon + (2^{n \cdot R} - 1) 2^{-n(I(X;Y)-3\epsilon)} \\ &\leq \epsilon + 2^{n \cdot R} 2^{-n(I(X;Y)-3\epsilon)} \\ &= \epsilon + 2^{-n(I(X;Y)-R-3\epsilon)} \end{aligned}$$

As long as $R < I(X;Y)$, one can choose n large enough so that $P[\texttt{error}] \leq 2\epsilon$.

This analysis upper bounds the (expected) average error probability for a random code $\mathcal{C}$. However, if this expected probability is low, there must be some specific code $\mathcal{C}^*$ that also has low average error probability.

Finally, in $\mathcal{C}^*$, we aim to bound the *maximal* error probability, i.e., the probability of error for the worst message. We can do so by noting that at least half of the messages w has error probability $\lambda_w(\mathcal{C}^*) \leq 4\epsilon$: if not, then the total error probability of these messages would already exceed $2^{n \cdot R} \cdot 2\epsilon$, contradicting the upper bound of 2ϵ to the average error probability. Thus, we can construct a better code by discarding the worst half of the codewords, and using the remaining $2^{n \cdot R - 1}$ codewords to construct a new code, with rate

$$\frac{\log(2^{n \cdot R - 1})}{n} = \frac{n \cdot R - 1}{n} = R - \frac{1}{n}$$

and maximal probability of error $\lambda^{(n)} \leq 4\epsilon$.

In the above proof, we implicitly assumed that $2^{n \cdot R}$ is an integer. You can try to redo the proof for the case when it is not: construct $\mathcal{C}$ as a $(\lceil 2^{n \cdot R} \rceil, n)$ code, and verify that the average probability of error $P[\texttt{error}]$ is still sufficiently small. Also compute a lower bound on the rate of the final code.

Noisy-Channel Theorem: Converse

In the previous section, we showed that any rate strictly below the channel capacity is achievable. Here, we show that one cannot do better: rates strictly above the channel capacity are not achievable. Specifically, codes with such rates suffer from non-negligible error probabilities.

Theorem: Shannon's noisy-channel coding theorem (converse)

On a discrete memoryless channel with capacity C, any code with rate $R > C$ has average probability of error $p_e^{(n)} \geq 1 - \frac{C}{R} - \frac{1}{nR}$.

Proof

For a code with rate $R > C$, let W be uniformly distributed over all possible messages, let X^n describe the encoding of the message (and the input to the channel), let Y^n describe the output of the channel, and $\hat{W}$ the decoding of that output.

The average probability of error, $p_e^{(n)}$, is equal to $P[W \neq \hat{W}]$, the probability that the original message differs from the decoded message. Note that $W \to X^n \to Y^n \to \hat{W}$ forms a Markov chain.

As a first step, we show that the mutual information between the message W and the channel output Y^n is upper bounded by $n \cdot C$, that is, there is a limit to the amount of information that can get through the channel. To see this, first observe that

$$\begin{aligned}
H(Y^n W) &= H(Y^n W) + H(X^n \mid Y^n W) && \text{(since } W \text{ determines } X \\
&= H(X^n Y^n W) && \text{(chain ru} \\
&= H(X^{n-1} Y^{n-1} W) + H(Y_n \mid X^n Y^{n-1} W) + H(X_n \mid X^{n-1} Y^{n-1} W) && \text{(chain ru} \\
&= H(X^{n-1} Y^{n-1} W) + H(Y_n \mid X^n Y^{n-1} W) && \text{(since } W \text{ determines } X \\
&= H(X^{n-1} Y^{n-1} W) + H(Y_n \mid X_n) && \text{(memoryle} \\
&= \dots && \text{(repe} \\
&= H(W) + \sum_{i=1}^{n} H(Y_i \mid X_i).
\end{aligned}$$

Therefore,

$$\begin{aligned}
I(W; Y^n) &= H(W) + H(Y^n) - H(Y^n W) && \text{(entropy diagram)} \\
&= H(Y^n) - \sum_{i=1}^{n} H(Y_i \mid X_i) && \text{(by the above derivation)} \\
&\leq \sum_{i=1}^{n} H(Y_i) - \sum_{i=1}^{n} H(Y_i \mid X_i) \\
&= \sum_{i=1}^{n} I(X_i; Y_i) \\
&\leq n \cdot C.
\end{aligned}$$

Now that we have established that $I(W; Y^n)$ is upper bounded by $n \cdot C$, we can show that the code with rate R induces a considerable error probability:

$$\begin{aligned} R &= \frac{\log |\mathcal{W}|}{n} \\ &= \frac{1}{n} H(W) \\ &= \frac{1}{n} (H(W \mid Y^n) + I(W; Y^n)) \\ &\leq \frac{1}{n} (H(W \mid Y^n) + n \cdot C) \\ &\leq \frac{1}{n} \Big(1 + P[W \neq \hat{W}] \cdot n \log |\mathcal{W}| + n \cdot C \Big) \\ &= \frac{1}{n} + P[W \neq \hat{W}] \cdot R + C, \end{aligned}$$

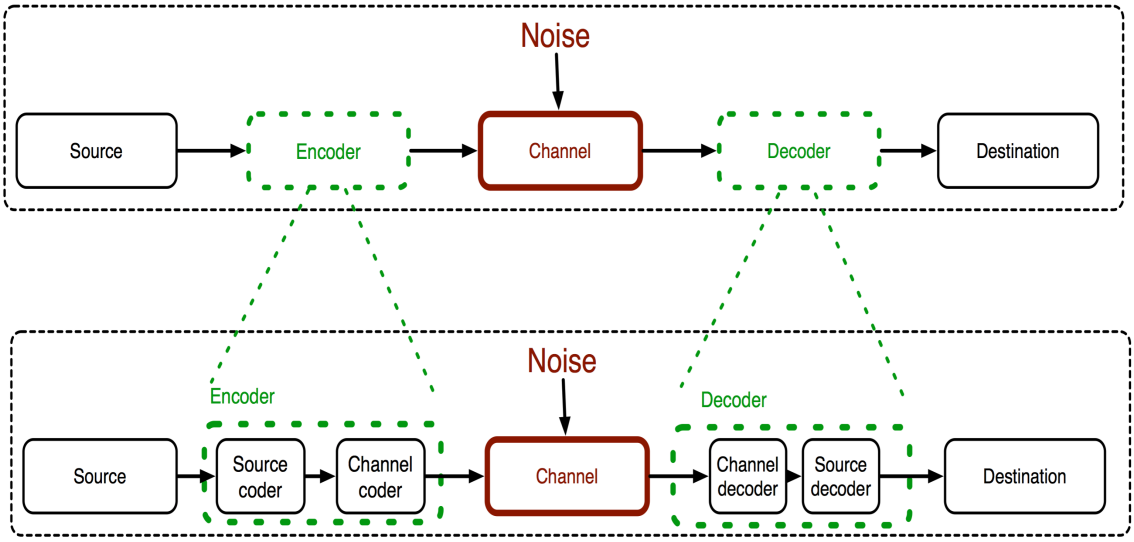
where the second inequality is an application of Fano's inequality. Dividing both sides by R and rearranging, we get the desired inequality:

$$p_e^{(n)} = P[W \neq \hat{W}] \geq 1 - \frac{C}{R} - \frac{1}{nR}.$$

This theorem shows that if Alice and Bob try to communicate using a code with a rate $R > C$, their probability of error will be bounded away from zero by a constant factor of $1 - \frac{C}{R}$ (for big n, the last term in the inequality becomes insignificant). This error probability worsens for a bigger difference between R and C.

Source-Channel Coding

In this final part of the course, we want to study the "ultimate problem" of transmitting a source of information over a noisy channel.



(Image credit: ECE 534 by Natasha Devroye, thanks a lot!)

Recall that a source can be modeled as P_V and a noisy channel by a conditional probability distribution $P_{Y|X}$. The problem of **source-channel coding** studies if there exists an encoding procedure mapping n iid samples V^n from the source P_V to channel inputs X^n, such that the resulting channel outputs Y^n can be decoded back to $\hat{V}^n$ in such a way that $P[V^n \neq \hat{V}^n]$ vanishes for $n \to \infty$.

One way of building such a source-channel encoder and decoder coder is sketched in the lower part of the figure above. We can use our knowledge from Module 03 of this course to build an optimal source coder that compresses the source information (with vanishing error) down to a rate $R > H(V)$ bits per source symbol. Then, we can use the results of this model to build a channel coder which will work with vanishing error as long as the rate $R < C$ is smaller than the capacity. From this intuition, it seems that requiring $H(V) < C$, i.e. that the entropy of the source we want to transmit is strictly smaller than the capacity of the channel, is a necessary and sufficient condition for transmitting a source over a channel.

created: 2019-10-21

We will see on the following pages that under certain conditions, that is indeed the case. However, let me try to convince (and confuse) you here that it is *a priori* not clear that separating the source and channel coder is such a good idea:

1. Recall the example from the very first video (starting at 2:31) of English text sent over an erasure channel. It seems that for this kind of source (English text) and channel (erasure channel), it works pretty well to simply not encode the text at all, and then try to recover (using some language model, in your brain or on a computer) from the erasures by filling in the missing letters. That is a very different source-channel coder than the one suggested above, which would first compress the text (say using zip) to get rid of redundancies, and then insert again some redundancy (using some error-correcting code) to survive the erasure channel.
2. Consider the cocktail-party effect. Our brains seems to be extremely good at extracting individual conversations in very noisy situations (such as at cocktail parties). You can experience that yourself by repeatedly watching the following video, trying to focus to understand either the left or right speaker:

   Again, in such a scenario, it seems rather wasteful to first compress the audio information to (basically) random noise, and then re-encoding it to survive the noisy environment, and then undo these steps to decode. Speaking English with a human voice already seems to be a very good encoder of information for this scenario, especially combined with a very powerful human-brain decoder.

Source-Channel Separation Theorem: Forward Direction

So far in this course, we have treated the 'encoding' of information from two different perspectives:

- Codes for compressing information (in order to achieve **efficient** communication). Shannon's source-coding theorem tells us that the minimal codeword length $\ell_{min}(P_{V^n})$ for encoding a input block from the source V^n (where all V_i are i.i.d. according to some distribution P_V) is lower bounded by $H(V^n) = nH(V)$. In other words, the rate for such a code, which is $R = \frac{\log|\mathcal{V}^n|}{n}$, is lower bounded by $H(V)$.
- Codes for protecting information (in order to achieve **low-error** communication). Shannon's channel-coding theorem tells us that in order to achieve an arbitrarily low error on the communication over a specific channel, it is necessary that the rate R is strictly upper bounded by the channel capacity C.

Combining these two perspectives, we can ask ourselves what happens in the 'sweet spot' where both $H(V) \leq R$ (as given by the first perspective) and $R < C$ (as given by the second perspective). Do codes with such R always exist? The answer turns out to be yes:

Theorem: Source-channel separation theorem (forward direction)

Let $V_1, V_2, \ldots, V_n$ be i.i.d. random variables (the source) distributed according to some P_V. Let $(\mathcal{X}, P_{Y|X}, \mathcal{Y})$ be a channel with capacity C. If $H(V) < C$, then there exists a source-channel code with error probability $P[\hat{V^n} \neq V^n] \to 0$ as $n \to \infty$.

Proof

The main idea is to construct our code in two steps: first, we optimally compress the source. Then, we apply an error-correcting code to that compression. Let $0 < \varepsilon < C - H(V)$. We will exhibit a code that has error probability at most ε. (This immediately implies the existence of such a code for $\varepsilon \geq C - H(V)$ as well.) For the first part (compression), consider the typical set $A_{\varepsilon/2}^{(n)}$ for the source P_V.

The typical set has size at most $2^{n(H(V)+\frac{\varepsilon}{2})}$, and, for large enough n, contains at least $1-\frac{\varepsilon}{2}$ of the probability mass of the source. Hence, we need $n(H(V)+\frac{\varepsilon}{2})$ bits to compress the source, with a loss (error) of at most $\frac{\varepsilon}{2}$.

For the second part (error-correction), we just learned that it is possible (for large enough n) to transmit with error less than $\frac{\varepsilon}{2}$, as long as $R<C$. Since we have compressed n source symbols into $n(H(V)+\frac{\varepsilon}{2})$ bits, the rate we want to achieve is $R=H(V)+\frac{\varepsilon}{2}$. By assumption that $\varepsilon<C-H(V)$, it follows that $R<C$, such a code indeed exists. Combining the two codes, each with error at most $\frac{\varepsilon}{2}$, and applying the union bound, we get

$$\begin{aligned} P[\hat{V}^n \neq V^n] &\leq P[V^n \notin A^{(n)}_{\varepsilon/2}] + P[\hat{V}^n \neq V^n | V^n \in A^{(n)}_{\varepsilon/2}] \\ &\leq \frac{\varepsilon}{2} + \frac{\varepsilon}{2} \\ &= \varepsilon. \end{aligned}$$

It is interesting to note that the i.i.d. assumption on the source can be relaxed, and the theorem actually holds for any finite-alphabet stochastic process satisfying the AEP and entropy rate $H(\{V_i\}) < C$.

Source-Channel Separation Theorem: Converse

Conversely, if a source V has entropy $H(V)$ that exceeds the channel capacity C, then it is impossible to transmit the source over the channel with arbitrarily small error.

Theorem: Source-channel separation theorem (converse)

Let $V_1, V_2, \dots, V_n$ be i.i.d. random variables (the source) distributed according to some P_V. Let $(\mathcal{X}, P_{Y|X}, \mathcal{Y})$ be a discrete memoryless channel with capacity C. If $H(V) > C$, then any source-channel code has an error probability that is bounded away from zero.

Proof

Assume the error probability $p_e := P[\hat{V}^n \neq V^n]$ *does* converge to zero as n goes to infinity. We show that $H(V) \leq C$, in order to prove the theorem by contraposition.

$$
\begin{aligned}
H(V) &= \frac{H(V^n)}{n} && (\text{since the source is i.i.d.}) \\
&= \frac{1}{n}(H(V^n|\hat{V}^n) + I(V^n;\hat{V}^n)) && (\text{by entropy diagrams}) \\
&\leq \frac{1}{n}(1 + p_e \log(|\mathcal{V}|^n) + I(V^n;\hat{V}^n)) && (\text{by Fano's inequality}) \\
&= \frac{1}{n} + p_e \log(|\mathcal{V}|) + \frac{1}{n} I(V^n;\hat{V}^n) \\
&\leq \frac{1}{n} + p_e \log(|\mathcal{V}|) + \frac{1}{n} I(X^n;Y^n) && (\text{by the data-processing inequality}) \\
&\leq \frac{1}{n} + p_e \log(|\mathcal{V}|) + C && (\text{by the lemma (1) on multiple channel uses}) \\
&\to 0 + 0 + C = C,
\end{aligned}
$$

as n goes to infinity.

(1) See here.

We have shown the source-channel separation theorem (the forward direction and its converse) for discrete, memoryless channels, and i.i.d. sources. It also holds for channels with feedback, and any sources that satisfy the asymptotic equipartition property.

created: 2019-10-21