# Information Theory

Muchang Bahng

Spring 2024

## Contents

First we establish some notation. Given a random variable $f : (\Omega, \mathcal{F}, \mathbb{P}) \to (A, \mathcal{A})$, $A$ is usually called the **alphabet** of $f$.

# 1 Entropy and Information

Let's motivate entropy. First, we want to quantitatively measure the "surprise" of an event $E$ happening in a probability space by assigning it a value $H(E)$. We want it to satisfy the following:

1. *Positive Surprisal.* $H(E) \geq 0$.
2. *No Surprisal from Sure Events.* $H(E) = 0$ iff $\mathbf{P}(E) = 1$.
3. *Additivity of Independent Events.* If $E_1$ and $E_2$ are independent events, then $H(E_1 \cap E_2) = H(E_1) + H(E_2)$.
4. *Continuity.* $H$ should be continuous, i.e. slight changes in probability correspond to slight changes in surprisal.

The third condition reminds us of the log function, and the fourth condition reminds us to do take advantage of continuity of measure, leading us to define the surprisal of an event as $-\log \mathbf{P}(E)$. Now we can define entropy as the expected surprisal of a random variable, which seems now more motivated. Intuitively, this represents the element of surprise of a certain data point, and distributions that have relatively sharp peaks will have lower entropy (since we expect most of the samples to come from the peaks) while uniform distributions have higher entropy.

## 1.1 Discrete Random Variables

Entropy was originally created by Shannon as part of his theory of communication. This is how the vast majority of students learn about entropy. Even though it is not as general, this definition is good to know.

**Definition 1.1 (Entropy in Discrete Setting)**

Let $(\Omega, \mathcal{F}, \mathbf{P})$ be a probability space.

1. *Paritition.* Let $\mathcal{Q} = \{Q_k\}_{k=1}^n$ be a finite partition of measurable subsets of $\Omega$. Then, the **entropy of partition** $\mathcal{Q}$ is defined
$$H_{\mathbf{P}}(\mathcal{Q}) = -\sum_{k=1}^{n} \mathbf{P}(Q_k) \log \mathbf{P}(Q_k) \tag{1}$$
2. *Random Variable.* Let $X$ be a random variable over $\Omega$ taking values in finite measure space $(A, 2^A)$. The **entropy of random variable** $X$ is defined
$$H_{\mathbf{P}}(X) \coloneqq -\sum_{a \in A} \mathbf{P}_X(\{a\}) \log \mathbf{P}_X(\{a\}) \tag{2}$$
$$= -\sum_{a \in A} \mathbf{P}(X^{-1}(a)) \log \mathbf{P}(X^{-1}(a)) = H_{\mathbf{P}}(\{X^{-1}(a)\}_{a \in A}) \tag{3}$$

where we define $0 \log 0 = 0$.[a]

[a] This is consistent with the limit.

The base of the logarithm differs: in mathematical theory, it is usually base $e$, but in signal processing, it is base 2 for bits.

**Definition 1.2 (Relative Entropy in Discrete Setting)**

Let $\mathbf{P}, \mathbf{Q}$ be two probability measures on $(\Omega, \mathcal{F})$. Given random variable $X$ taking values in finite

alphabet $(A, 2^A)$, the **relative entropy of $X$ with measure $\mathbf{P}$ w.r.t. reference measure $\mathbf{Q}$** is

$$H_{\mathbf{P}||\mathbf{Q}}(X) \coloneqq \begin{cases} \sum_{a\in A} \mathbf{P}_X(\{a\}) \log \frac{\mathbf{P}_X(\{a\})}{\mathbf{Q}_X(\{a\})} & \text{if } \mathbf{P}_X \ll \mathbf{Q}_X \\ +\infty & \text{if else} \end{cases} \tag{4}$$

where $\mathbf{P}_X, \mathbf{Q}_X$ are the induced measures on $\mathcal{A}$.

**Definition 1.3 (Divergence)**

Given a finite measure space $(\Omega, 2^\Omega)$, with two probability measures $\mathbf{P}, \mathbf{Q}$.

$$D(\mathbf{P} \mid\mid \mathbf{Q}) = \sum_{w\in\Omega} \mathbf{P}(\{\omega\}) \log \frac{\mathbf{P}(\{\omega\})}{\mathbf{Q}(\{\omega\})} \tag{5}$$

Note that $H_{\mathbb{P}||\mathbb{Q}}(X) = D(\mathbf{P}_X||\mathbf{Q}_X)$.

**Lemma 1.1 (Divergence is Nonnegative)**

Given any probability measures $\mathbf{P}, \mathbf{Q}$ on a finite measure space $(\Omega, 2^\Omega)$, we have

$$D(\mathbf{P}||\mathbf{Q}) \geq 0 \tag{6}$$

with equality iff $\mathbf{P} = \mathbf{Q}$.

*Proof.*

## 1.2 Relative Entropy

**Definition 1.4 (Relative Entropy)**

The **relative entropy**, also called the **(Kullback-Leibler) divergence**, of two probability measures $\mathbf{P}, \mathbf{Q}$ is defined in the following ways:

1. *Normal Definition.* Given that $\frac{d\mathbf{P}}{d\mathbf{Q}}$ is the Radon-Nikodym derivative.

$$D(\mathbf{P}||\mathbf{Q}) \coloneqq \begin{cases} \mathbb{E}_{\mathbf{P}}\left[\log \frac{d\mathbf{P}}{d\mathbf{Q}}\right] = \int \log \frac{d\mathbf{P}}{d\mathbf{Q}}\, d\mathbf{P} & \text{if } \mathbf{P} \ll \mathbf{Q} \\ +\infty & \text{if else} \end{cases} \tag{7}$$

2. *Variational.* Given that $D$ is the divergence of discrete probability measures,

$$D(\mathbf{P}||\mathbf{Q}) \coloneqq \sup_{X \text{ simple}} D(\mathbf{P}_X||\mathbf{Q}_X) \tag{8}$$

It is a measure of how one probability distribution is different from a second, reference probability distribution. A relative entropy of 0 indicates that $p$ and $q$ are identical. It is useful to interpret this measure as a "distance" between two distributions, but it is not a formal metric because it is not symmetric and does not satisfy the triangle inequality.

**Lemma 1.2**

The divergence is always nonnegative.

*Proof.* By Jensen's inequality.

**Example 1.1**

If $\mu, \nu$ are both continuous distributions, we have

$$\begin{aligned} \mathrm{KL}(p||q) &\coloneqq -\int p(x)\, \ln q(x)\, dx - \left( -\int p(x)\, \ln p(x)\, dx \right) && (9) \\ &= -\int p(x)\, \ln \left( \frac{q(x)}{p(x)} \right) dx && (10) \end{aligned}$$

## 1.3 Mutual Information

**Definition 1.5 (Mutual Information)**

Consider two random variables $X, Y$ with respective probability measures $\mu_X, \mu_Y$, let $\mu_{X\times Y}$ be their joint distribution and $\mu_X \times \mu_Y$ be their product measure. The **mutual information** of $X$ and $Y$ is defined

$$I(X;Y) \coloneqq D(\mu_{X\times Y}||\mu_X \times \mu_Y) \tag{11}$$

**Theorem 1.3 (Mutual Information and Entropy)**

The mutual information $I(X;Y)$ is the reduction in the uncertainty of $X$ due to the knowledge of $Y$.

$$I(X;Y) = H(X) - H(X|Y) \tag{12}$$

It follows that

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X) \\ &= H(X) + H(Y) - H(X,Y) \\ I(X;Y) &= I(Y;X) \\ I(X;X) &= H(X) \end{aligned}$$

*Proof.* We can write

$$\begin{aligned} I(X;Y) &= \sum_{x,y} p(x,y)\ \log \frac{p(x,y)}{p(x)p(y)} \\ &= \sum_{x,y} p(x,y) \log \frac{p(x|y)}{p(x)} \\ &= -\sum_{x,y} p(x,y) \log p(x) + \sum_{x,y} p(x,y) \log p(x|y) \\ &= -\sum_{x} p(x) \log p(x) - \left( -\sum_{x,y} p(x,y) \log p(x|y) \right) \\ &= H(X) - H(X|Y) \end{aligned}$$

By symmetry, we have

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

Thus, $X$ says as much about $Y$ as $Y$ says about $X$. Since $H(X,Y) = H(X) + H(Y|X)$, we have

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

That is, the mutual information of a random variable with itself is the entropy of the random variable. This is the reason that entropy is sometimes referred to as *self-information*.

**Example 1.2**

For the joint distribution in the previous example, the mutual information is

$$I(X;Y) = H(Y) - H(Y|X) = 2 - \frac{13}{8} = \frac{3}{8} \text{ bits}$$

**Definition 1.6**

The **conditional mutual information** of random variables $X, Y, Z$ is the reduction in the uncertainty of $X$ due to knowledge of $Y$ when $Z$ is given.

$$\begin{aligned} I(X;Y|Z) &= H(X|Z) - H(X|Y,Z) \\ &= E_{p(x,y,z)} \left( \log \frac{p(X,Y|Z)}{p(X|Z)p(Y|Z)} \right) \end{aligned}$$

Mutual information also satisfies a chain rule.

**Theorem 1.4 (Chain rule for information)**

$$I(X_1, X_2, ..., X_n; Y) = \sum_{i=1}^{n} I(X_i; Y | X_{i-1}, ..., X_1)$$

*Proof.*

$$\begin{aligned} I(X_1, ..., X_n; Y) &= H(X_1, ..., X_n) - H(X_1, ..., X_n|Y) \\ &= \sum_{i=1}^{n} H(X_i|X_{i-1}, ..., X_1) - \sum_{i=1}^{n} H(X_i|X_{i-1}, ..., X_1; Y) \\ &= \sum_{i=1}^{n} I(X_i; Y|X_1, X_2, ..., X_{i-1}) \end{aligned}$$

## 1.4 Entropy

**Definition 1.7 (Entropy)**

The **entropy** of a random variable $X$ is defined

$$H(X) \coloneqq I(X; X) \tag{13}$$

## 1.5 Differential Entropy

**Definition 1.8 (Differential Entropy)**

For a continuous random vector, the **differential entropy** is defined

$$H[X] = -\int p(x) \ln p(x)\, dx \tag{14}$$

## 1.6 Exercises

**Exercise 1.1 (63 Puzzle)**

I am thinking of an integer $0 \leq x \leq 63$. You must identify this $x$ by asking if it is at least a number $y$. How do you get it in the minimum number of questions?

*Proof.* The answer is clearly binary search, which gets it done in $\log_2 64 = 6$ questions. More specifically, we can come up with the following predetermined questions which each gives the $i$th binary digit of the solution.

1. $C_1 : x \bmod 64 \geq 32$?
2. $C_2 : x \bmod 32 \geq 16$?
3. $C_3 : x \bmod 16 \geq 8$?
4. $C_4 : x \bmod 8 \geq 4$?
5. $C_5 : x \bmod 4 \geq 2$?
6. $C_6 : x \bmod 2 \geq 1$?

Note that if we are assuming a uniform distribution, this is the strategy that maximize stepwise entropy, since the outcome of each question has an equal probability of being 0 or 1, leading to 1 bit of information (and not less since all these random variables $C_i$ are independent) This is indeed exactly how much information about the binary expansion of $x$ we get. After 6 questions, the total information content was 6 bits.

From this, we can claim something.

**Lemma 1.5**

All outcomes of a random variable $X$ from a set of size $S$ can be communicated in $\lceil \log_2 |S| \rceil$ bits.

Let's think more about what information means with another game.

**Example 1.3 (Submarines)**

You are playing battleship on a $8 \times 8$ grid, but there is one $1 \times 1$ submarine that you are trying to hit. Say you choose some square and hit it.

1. You don't hit it, which happens with probability 63/64. You then get $\log_2(64/63) \approx 0.0227$ bits of information.
2. You fire again and don't hit, which happens with probability 62/63. You get $\log_2(63/62) \approx 0.0230$ bits of information. The total information gained is 0.04560.
3. You keep firing off at squares. You obviously don't want to fire at an already hit square since the probability that you don't hit is 1, so no information is gained.
4. If you keep firing and don't hit after 32 tries, then you have gained a total information of
$$\log_2 \frac{64}{63} + \ldots + \log_2 \frac{33}{32} = 1 \tag{15}$$
bit. This is similar to getting 1 bit from the first step of binary search, which is consistent with our intuition.
5. Let's keep firing and say on the 35th hit, we actually hit the submarine. Then our total information content is
$$\log_2 \frac{64}{63} + \ldots + \log_2 \frac{33}{32} + \ldots + \log_2 \frac{21}{20} + \log_2 \frac{20}{1} = 6 \tag{16}$$
We have then acquired 6 bits of information (around 4.3 bits for the hit) and gotten all possible information we can get from the grid.

**Lemma 1.6 (Approximation of Binomial Distribution)**

We can approximate
$$\binom{n}{k} = \log \frac{n!}{k!(n-k)!} \approx nH_2\left(\frac{p}{n}\right) \tag{17}$$

**Exercise 1.2 (Bent Coin Lottery)**

A coin with $p = 0.1$ is tossed 1000 times to get a random vector $\mathbf{x} \in \{0,1\}^{1000}$. You can buy any of the $2^N$ possible tickets for \$1 each, before the coin tossing. If you own the correct ticket, you get a lot of money.

1. If you are forced to buy one ticket, which ticket would you buy?
2. To have a 99% change of winning at a lowest possible cost, which tickets should you buy?
3. And how many tickets is that? Express it in the form of $2^n$.

*Solution.* Let's go through them.[a]

1. Even though the expected number of 1's is 100, the all 0 ticket would be the most likely outcome.
2. From the previous problem, we can intuit that we should buy all the tickets with zero 1s, then one 1s, then two 1s, and so on until some threshold $r$ where the probability is 99%. By CLT, we can approximate this to be normal with mean 100 and standard deviation $\sqrt{1000 \cdot 0.1 \cdot 0.9} \approx 9.5$, and therefore a z-score of about 2.3 will give us $100 + 9.5 * 2.3 = 121.85$ tosses with a 99% chance of winning.

3. To find out how many tickets this is, we compute

$$1 + \binom{1000}{1} + \binom{1000}{2} + \ldots + \binom{1000}{123} \tag{18}$$

the rightmost is the dominant term, and we use the approximation to get it approximately $2^{530}$ tickets.

[a] Thanks to Thomas Kidane for pointing out my initial mistakes in the solution.

Therefore, we have essentially "compressed" the set of all $2^{1000}$ tickets up to 99% probability of hitting, into a set of approximately $2^{530}$ tickets, called the **typical set**, which can be encoded in 530 bits. Therefore,

1. the compressor takes the typical set of tickets and creates a bijection into a second set of 530 bit long strings.
2. The decompressor just undos this bijection from the typical set back into the 1000 bit long strings.

In a general case lottery with $n$-length strings and a probability of $p$, we can compute that you will need approximately

$$\binom{n}{fn + 2.3\sqrt{Nf(1-f)}} \approx 2^{NH_2(p)+\epsilon} \tag{19}$$

where $\epsilon$ is a small term that scales with $\sqrt{n}$. We see a certain pattern that coincides with the source coding theorem on how well we can compress a certain set that scales with some probability. Note that this depends on precisely defining the typical set.

**Definition 1.9 (Typical Set)**

When a source $X$ produces $N$ independent outcomes

$$\mathbf{x} = x_1, x_2, \ldots, x_n \tag{20}$$

This string is very likely to be in a **typical set** consisting of $\sim 2^{nH(X)}$ outcomes all of which have a probability of $\sim 2^{-nH(X)}$.

**Theorem 1.7 (Source Coding Theorem)**

$N$ outcomes from a source $X$ can be compressed into roughly $NH(X)$ bits.

We have hinted at the fact through Shannon's noisy encoding theorem that there is an optimal way to add redundancies to compress some input. Given a string of random variables $X_1, \ldots, X_n$ generated iid from a Bernoulli($p$) distribution, we want to start to formalize this by introducing a metric to measure the information content of this stochastic process. We motivate the necessity of such a measure using general probability measures and then focus on the discrete case.

## 1.7 Discrete Random Variables

In Shannon's famous paper [1], he talks first on discrete channels, focusing on examples of transmitting languages through n-gram models as "higher order approximations" of language.[1] This is an ergodic Markov chain with some stationary distribution.

[1] In fact, this is where n-gram models were first referenced.

**Theorem 1.8 (Bounds on Entropy)**

$H$ is bounded by 0 and 1, attaining its minimum if and only if all the $p_i$ but one are 0. It attains its maximum if $p$ is uniform.

**Theorem 1.9 (Entropy of Independent Events)**

If $E_1$ and $E_2$ are independent events, then $H(E_1 \cap E_2) = H(E_1) + H(E_2)$. The information from two independent events is the sum of their informations since information gain from one does not increase information from another independent variable.

**Example 1.4 (Bits)**

Given $X \sim \text{Bernoulli}(p)$, if we observe a value of 1, then we have received $\log_2\left(\frac{1}{p}\right)$ bits of information.

Now Shannon's claim is that this information content is the optimal encoding length that we should aim for. For example, given $p = 0.9$, then a 0 has 3.32 bits of information content and a 1 has 0.15 bits. This means that 0's, which occur infrequently, should be encoded with longer strings and 1 with shorter strings.

**Exercise 1.3 (Weighing Problem)**

You are given 12 balls, all equal in weight except for one that is either heavier or lighter. Design a strategy to determine which is the odd ball *and* whether it is heavier or lighter in as few uses of the balance as possible.

*Proof.* We can tackle this by looking at the first action. We can choose to weigh $n$ vs $n$ balls for $n = 1, \dots, 6$. Shannon would advise you to choose such that we maximize our entropy, or expected information gain. Let's go through them one at a time. Our three outcomes for all scenarios are $A$ (left is lighter), $B$ (both equal), and $C$ (right is lighter).

1. 6 v 6. The probability distribution is $(A, B, C) = (1/2, 0, 1/2)$ and so the entropy is $H = 1$ bit.
2. 5 v 5. The distribution is $(5/12, 1/6, 5/12)$ giving us $H = 1.48$ bits.
3. 4 v 4. The distribution is $(1/3, 1/3, 1/3)$ giving us $H = 1.58$ bits.
4. We go on.

We already know that entropy must be maximized in the uniform distribution, so it is best to choose 4 v 4. This is indeed the correct first step. As for the next step. Let's think about what to do in each of the events.

1. $A$. This means that there are four $H$'s (possibly heavies), four $L$'s, and four $G$'s (possibly good). We are left with 8 balls and we want to maximize the entropy. It turns out that if we measures HHL vs HHL, then the events turn out to have distribution $(3/8, 2/8/3/8)$, which is quite uniform.
2. $B$. We have eight $G$'s, so to maximize the entropy we can weigh one against another, which has a distribution of $(1/4, 1/2, 1/4)$.
3. $C$. By symmetry, we use the same method as $A$.

We just continue this process which is a stepwise optimization of entropy. It turns out that we just need 3 steps.

## 1.8 Joint and Conditional Entropy

The naturalness of the definition of joint entropy and conditional entropy is exhibited by the fact that the entropy of a pair of random variables is the entropy of one plus the conditional entropy of the other.

**Definition 1.10 (Joint, Conditional Entropy)**

We can define the joint entropy and conditional entropy between two discrete random variables $X, Y$ as

$$H(X,Y) = \mathbb{E}_{X\times Y}[-\log p(x,y)] = \sum_{x,y\in\mathcal{X},\mathcal{Y}} p(x,y)\cdot -\log p(x,y)$$
$$H(X \mid Y) = \mathbb{E}_{X\times Y}[-\log p(x \mid y)] = \sum_{x,y\in\mathcal{X},\mathcal{Y}} p(x,y)\cdot -\log p(x \mid y)$$

**Example 1.5 (Dice Rolls)**

Let $(X, Y)$ have the following joint distribution:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | $\frac{1}{8}$ | $\frac{1}{16}$ | $\frac{1}{32}$ | $\frac{1}{32}$ |
| 2 | $\frac{1}{16}$ | $\frac{1}{8}$ | $\frac{1}{32}$ | $\frac{1}{32}$ |
| 3 | $\frac{1}{16}$ | $\frac{1}{16}$ | $\frac{1}{16}$ | $\frac{1}{16}$ |
| 4 | $\frac{1}{4}$ | 0 | 0 | 0 |

The marginal distribution of $X$ is $\left(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}\right)$ and the marginal distribution of $Y$ is $\left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right)$, meaning that

$$H(X) = \frac{7}{4} \text{ bits}, \; H(Y) = 2 \text{ bits}$$

Also,

$$\begin{aligned} H(X|Y) &= \sum_{i=1}^{4} p(Y=i)\, H(X|Y=i) \\ &= \frac{1}{4}H\left(\frac{1}{2},\frac{1}{4},\frac{1}{8},\frac{1}{8}\right) + \frac{1}{4}H\left(\frac{1}{4},\frac{1}{2},\frac{1}{8},\frac{1}{8}\right) \\ &\quad + \frac{1}{4}H\left(\frac{1}{4},\frac{1}{4},\frac{1}{4},\frac{1}{4}\right) + \frac{1}{4}H(1,0,0,0) \\ &= \frac{1}{4}\cdot\frac{7}{4} + \frac{1}{4}\cdot\frac{7}{4} + \frac{1}{4}\cdot 2 + \frac{1}{4}\cdot 0 = \frac{11}{8} \text{ bits} \end{aligned}$$

Similarly, $H(Y|X) = \frac{13}{8}$ bits and $H(X,Y) = \frac{27}{8}$ bits.

**Theorem 1.10 (Joint Entropy)**

The uncertainty of a joint event is less than or equal to the sum of the individual uncertainties, with equality achieved only if the events are independent.

$$H(X,Y) \leq H(X) + H(Y) \tag{21}$$

Another property is that any change towards "equalization" of the probabilities $p_i$ increases $H$. Since we don't have a method of measuring how close to the uniform distribution, we will return back to this after defining the KL divergence.

**Theorem 1.11 (Chain Rule of Entropy)**

The joint entropy is the entropy of $X$ plus the conditional entropy of $Y$ given $X$.

$$H(X,Y) = H(X) + H(Y \mid X) = H(Y) + H(X \mid Y) \tag{22}$$

*Proof.* By linearity of expectation,

$$\begin{aligned} H(X) + H(Y|X) &= -\mathbb{E}\Big( \log \frac{1}{p(X)} \Big) - \mathbb{E}\big( \log p(Y|X) \big) \\ &= -\mathbb{E}\big( \log p(X) + \log p(Y|X) \big) \\ &= -\mathbb{E}\big( \log p(X)p(Y|X) \big) \\ &= -\mathbb{E}\big( \log p(X,Y) \big) = H(X,Y) \end{aligned}$$

**Corollary 1.12 (Chain Rule for Entropy)**

Let $X_1, X_2, ..., X_n$ be drawn according to $p(x_1, x_2, ..., x_n)$. Then

$$H(X_1, X_2, ..., X_n) = \sum_{i=1}^{n} H(X_i | X_{i-1}, ..., X_1) \tag{23}$$

**Theorem 1.13 (Conditioning Never Decreases Uncertainty)**

Since

$$H(X) + H(Y) \geq H(X,Y) = H(X) + H(Y \mid X) \tag{24}$$

we have $H(Y) \geq H(Y \mid X)$. That is, the uncertainty of $Y$ is never increased by the knowledge of $X$, which is expressed as

$$H(X \mid Y) \leq H(X) \tag{25}$$

with equality is $X, Y$ are independent. Indeed, the "uncertainty" of $X$ would decrease if we had any knowledge about a (potentially nonindependent) distribution $Y$.

In fact, the amount of uncertainty that decreases when conditioning has a well known name.

**Definition 1.11 (Mutual Information)**

The **mutual information** between random variables $X, Y$ is the decrease in entropy when we condition $X$ by $Y$.

$$I(X;Y) = H(X) - H(X \mid Y) = H(Y) - H(Y \mid X) \tag{26}$$

This can be conditioned on another random variable $Z$.

$$I(X;Y \mid Z) = H(X \mid Z) - H(X \mid Y, Z) = H(Y \mid Z) - H(Y \mid X, Z) \tag{27}$$

Therefore, we can interpret $I(X;Y)$ as the partial information you learn about $X$ from knowing $Y$. The entropy also demonstrates the average length (if base is 2) number of bits required to transmit the state of a random variable.

**Theorem 1.14**

From simple substitution, we can derive

$$H(X,Y) = H(X \mid Y) + H(Y \mid X) + I(X;Y) \tag{28}$$

Unlike entropy, conditioning the mutual information on a third variable can have either a hiding or revealing effect. It can *both* be the case that[2]

1. $I(X;Y \mid Z) > I(X;Y)$ happens when $X$ and $Y$ both are causes of some common effect $Z$, i.e. if you know $Z$ has happened, then $X$ and $Y$ are more dependent than before. For example, a car's engine fails to start (event $Z$), it may be because of either blocked fuel pump ($X$) or that the battery is dead ($Y$). Normally, $X$ and $Y$ are independent, so $I(X;Y) = 0$, but if the engine doesn't start, they suddenly become very dependent, since now you can look at the battery ($Y$) and from that conclude the status of the pump ($X$) with much more confidence, making $I(X;Y \mid Z) > 0$. [3]
2. $I(X;Y) > I(X;Y \mid Z)$ happens when $Z$ is the cause of both $X$ and $Y$. For example, if clouds ($Z$) always cause rain ($X$) and blocks the sun, ($Y$), then we know that $I(X;Y \mid Z) = 0$ since $Z$ already tells us everything about $X$ and $Y$, so $Y$ does not tell us anything more about $X$. But if we only observe whether the sun is blocked, this only tells us partially about whether it is rainy (may or may not be due to clouds or some other factor), making $I(X;Y) > 0$ due to some correlation revealed.

[2] Read the Wikipedia article on *Interaction Information*.

[3] Another example is given independent Bernoullis $X, Y$, with $Z = X \oplus Y \pmod 2$, we can clearly see that $I(X;Y) = H(X) - H(X \mid Y) = 0$ since they are independent and so $Y$ does not give any information about $X$. However, if we condition on $Z$ further, this gives complete information on $X$. This is quite intuitive since you would know more about $X$ from knowing both $Y$ and $Z$ rather than just knowing $Y$.

# 2 Channels and Codes

## 2.1 Channels

In a *communication system*, we have a *transmitter* and *receiver*, with *signals* going through a *channel*. Let's briefly define what these terms are, which are pretty much taken verbatim from Shannon's famous paper [1].

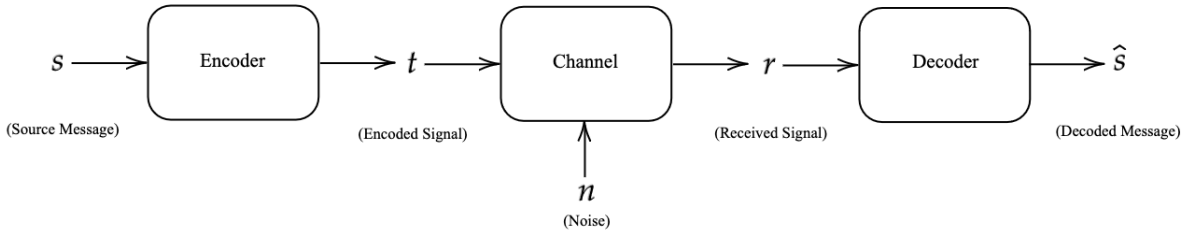



Figure 1: A channel diagram.

**Definition 2.1 (Information Source)**

An **information source** produces a message or sequence of messages to be communicated to the receiving terminal.

**Definition 2.2 (Encoder)**

A **transmitter**, or **encoder**, operator on the message in some way to produce a signal suitable for transmission over the channel.

**Definition 2.3 (Channel)**

The **channel** is the medium used to transmit the signal from the encoder to the decoder. Some examples of channels are:

1. A copper wire is a channel connecting one phone to another phone.
2. Air is a channel connecting your voice to another's ear.
3. Vacuum is a channel connecting an antenna on earth to the Mars rover.

**Definition 2.4 (Decoder)**

The **decoder**, or the **receiver** performs the inverse operation of that done by the transmitter, reconstructing the message from the signal.

**Definition 2.5 (Destination)**

The **destination** is the person (or thing) for whom the message is intended.

All the channels have the property that the received signal is maybe similar, but not identical, to the transmitted signal. This noise is not preferable, and we would ideally like to have perfect communications systems. To reduce this noise, we can improve physical systems (e.g. better insulation in copper wires) or we can improve our systems, such as our encoding/decoding schemes.

**Example 2.1 (Binary Symmetric Channel)**

Given a 1-bit input $x$, there is a certain probability $p$ such that the input is flipped.[a] This can be sometimes seen in practical applications, e.g. the salt-and-pepper noise in images.

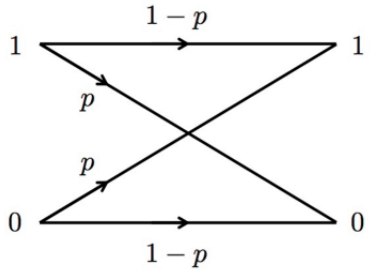



Figure 2: A simple example of noise.

[a] In 2014 disk drives, the standard was that $p$ should not be greater than $10^{-18}$.

## 2.2 Coding Schemes

To reduce the probability of $\hat{s} \neq s$, we can devise many schemes of the encoder and decoder. Depending on how much additional information we add, our channel throughput, or **rate**, becomes lower.

**Definition 2.6 (Parity Encoding)**

Given a string of bits, we can simply add a parity bit.

$$\text{encoder}(x_1, x_2, \dots, x_n) = x_1, \dots, x_n, (x_1 \oplus \dots \oplus x_n) \tag{29}$$

This has a rate of $n/(n+1)$.

**Definition 2.7 (Repetition)**

The encoder can just repeat each bit $k$ times, which we will denote as $R_k$.

$$\text{encoder}(x_1, \dots, x_n) = x_1, x_1, x_1, x_2, \dots, x_n \tag{30}$$

For example, with $k = 3$ we have

```
s = 01101
t = 000 111 111 000 111
n = 000 100 000 101 000
r = 000 011 111 101 111
```

The decoder then can take the best of 3 to get `01111`. Note that the second bit had a flip but was fixed, but the second to last bit was an error. We can then compute the probability of these errors with basic computations.[a] This has a rate of $1/k$.

[a] It turns out that we need $k = 61$ to get a probability of error below $10^{-15}$.

We can already predict that these encoding schemes can get quite sophisticated. Here's another one.

**Definition 2.8 (7, 4 Hamming Code)**

Given an input string of bits $\mathbf{s}$, we divide it up into sequences of 4.

$$\mathbf{s}_{i:i+4} = (s_i, s_{i+1}, s_{i+2}, s_{i+3}) \tag{31}$$

Then we can place them in a Venn diagram as shown below and fill out the rest of the three empty spots such that the parity within each circle is 0.

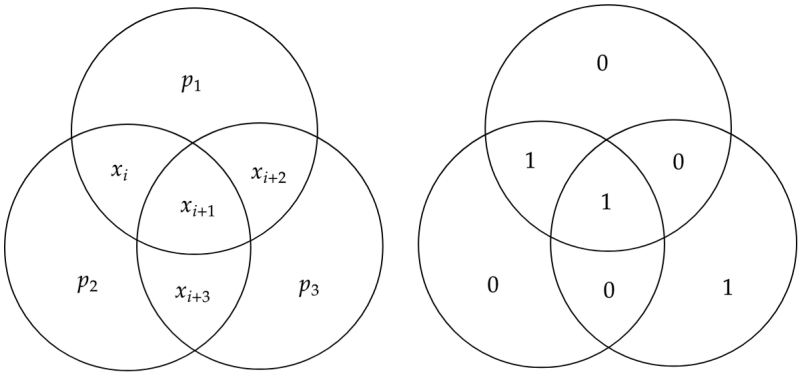



Figure 3: (7, 4) hamming code visual with example on the right.

This gives us the encoder.

$$\text{encoder}(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3, x_4, p_1, p_2, p_3) \tag{32}$$

As for the decoder, we can fill up the Venn diagram with the received bits $r_1, \dots, r_7$ and then look at the minimum number of bits needed to flip to achieve the same rules we had to fill the inputs out in the Venn diagram. Given any combination of circles that have parity 1, we can then flip exactly one of the $r_:$ to satisfy the rules again (i.e. find the bit that is outside all the valid circles and inside all the invalid circles). This has a rate of 4/7.

**Theorem 2.1 (Conditions for Detection and Correction)**

The (7,4) Hamming code can correct an input if up to 1 bit is flipped in each sequence of 4 bits, but if there are more than 1 bit flip, the decoded sequence will be incorrect.

More specifically, the probability of a block error is $21p^2$ on the most significant order and a bit error is $9p^2$.

If we look at these different algorithms and plot their rate vs probability of error, we can see some sort of dependency.

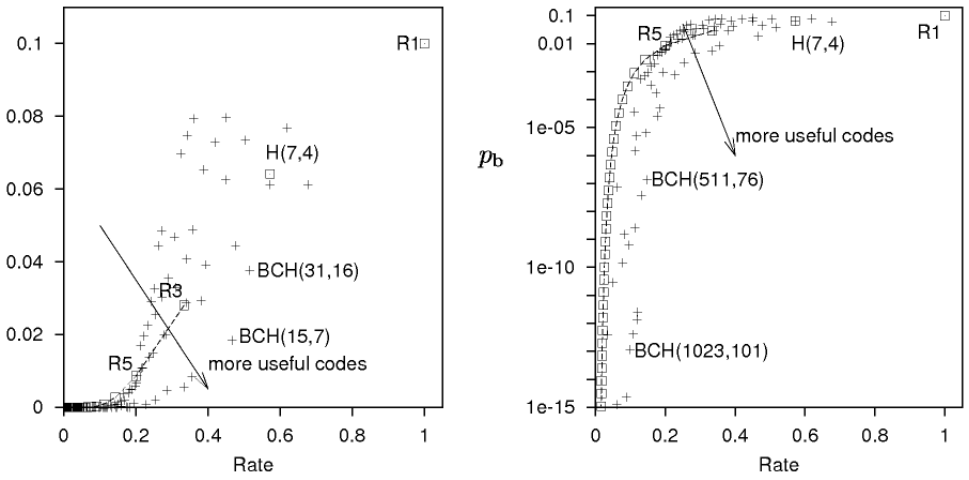



Figure 4: The rate of an encoding/decoding scheme vs probability of bit error.

It was reasonable to assume that we can make schemes that "hit" the upper-left portion of the left graph, i.e. we can make schemes that have a low rate (lots of repetition and such) yet still have a low probability of error. The question was how well we can reach the bottom-right corner containing the more useful codes. The general consensus assumed that as the probability of error goes to 0, the rate must also tend towards 0, and so we had a boundary that intersected through the origin that separated achievable and non-achievable schemes. However, Claude Shannon remarkably proved that this was not the case, through his *noisy-channel coding theorem*. Rather, we can achieve arbitrarily low probabilities without having to go below some non-zero rate, i.e. this boundary crosses the x-axis at some positive number $C$.

**Definition 2.9 (Capacity)**

$C$ is the **capacity** of the channel.

**Theorem 2.2 (Capacity of Binary Switch Channel)**

The capacity of the BSC with flip probability $f$ is

$$C_{BSC,f} = 1 - H(X), \quad X \sim \text{Bernoulli}(f) \tag{33}$$

This means that rather than needing 61 times our input to get past $10^{-15}$ error in the BSC with $f = 0.1$ (which we derive through repetition), we only need 2 disk drives, which is amazing.

**Definition 2.10 (Symbol Code)**

Let $X$ be a discrete random variable over some finite **alphabet** $\mathcal{S}$ with probability measure $\mathbb{P}$. A **symbol code** is a map $C : \mathcal{S} \to \{0,1\}^*$ of this **ensemble**. It maps the string

$$x_1, x_2, \dots, x_N \mapsto C(x_1), C(x_2), \dots, C(X_N) \tag{34}$$

It should satisfy the properties:

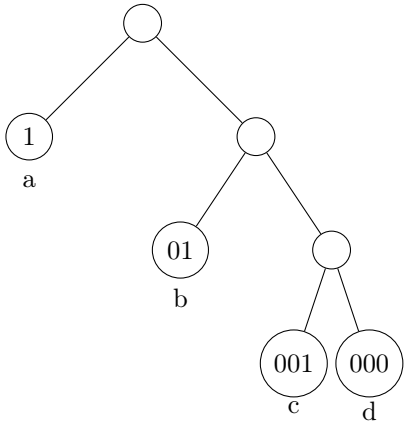
1. Every encoded string should be uniquely decodable.
2. It should be easy to decode in some sense.

3. The expected length
$$\mathbb{E}_{\mathbb{P}}[\ell(C)] = \sum_{s \in \mathcal{S}} \ell(C(s))\mathbb{P}(s) \tag{35}$$
of the encoded symbol should be small.

**Example 2.2 (Simple Code)**

Let's try to create symbol codes for $\mathcal{S} = \{a, b, c, d\}$ with probabilities $\{1/2, 1/4, 1/8, 1/8\}$.

1. The most obvious one is
$$C(s) = \begin{cases} 1000 & s = a \\ 0100 & s = b \\ 0010 & s = c \\ 0001 & s = d \end{cases} \tag{36}$$
with an expected length of 4.
2. We can perhaps shorten this by realizing that the trailing zeros are not needed.
$$C(s) = \begin{cases} 1 & s = a \\ 01 & s = b \\ 001 & s = c \\ 0001 & s = d \end{cases} \tag{37}$$
It does have an expected length of $1\frac{7}{8}$.
3. This one is not a valid scheme since 10010 can be decoded into $dc$ or $abd$.
$$C(s) = \begin{cases} 1 & s = a \\ 00 & s = b \\ 010 & s = c \\ 10 & s = d \end{cases} \tag{38}$$
It does have an expected length of $1\frac{5}{8}$.
4. Since we have 4 characters, we can just encode into a constant 2-bit string.
$$C(s) = \begin{cases} 00 & s = a \\ 01 & s = b \\ 10 & s = c \\ 11 & s = d \end{cases} \tag{39}$$
It does have an expected length of 2.
5. We can also see that if we have three 0s, then the next character must be a 1, so this is repetitive.
$$C(s) = \begin{cases} 1 & s = a \\ 01 & s = b \\ 001 & s = c \\ 000 & s = d \end{cases} \tag{40}$$
The expected length is $1\frac{3}{4}$, which turns out to be entropy of this probability distribution. We can




visualize this using a binary tree.

**Definition 2.11 (Prefix Code)**

Note that all of these encodings except for the nonvalid scheme has the property that no encoding of a character is a prefix of another character. A scheme with this property is called a **prefix code**.

**Example 2.3**

By modifying the best scheme so far by swapping the 0s and 1s, we have

$$C(s) = \begin{cases} 1 & s = a \\ 10 & s = b \\ 100 & s = c \\ 000 & s = d \end{cases} \tag{41}$$

But this is not a prefix code, though it is a valid code (uniquely decodable since its symmetric counterpart is uniquely decodable). For example, we can sequentially decode the string

$$1000000\ldots \tag{42}$$

since we don't know where the 0s end. While we may be able to decode this if we knew the length, it isn't really easy to decode.

The right intuition as this point is to give characters with large probabilities a short codeword and low ones longer ones. However, this is also not true.

**Example 2.4**

Let's have the same alphabet but now slightly perturb the probabilities

$$\mathbb{P}(a) = \frac{1}{4} + \epsilon, \mathbb{P}(b) = \frac{1}{4} + \frac{\epsilon}{2}, \mathbb{P}(c) = \frac{1}{4} - \frac{\epsilon}{2}, \mathbb{P}(d) = \frac{1}{4} - \epsilon, \tag{43}$$

Then our prefix coding would be

$$C(s) = \begin{cases} 1 & s = a \\ 01 & s = b \\ 001 & s = c \\ 000 & s = d \end{cases} \tag{44}$$

which still has an expected length of 2.25, which is not enough to beat just the regular 2-bit encoding of each word.

Here is a better system of thinking about this. If all codewords have length $l$, then the number of codewords that we can make is $2^l$. Then we can think of each codeword of length $l$ having a "cost" of $2^{-l}$ in our *codeword supermarket*.

<table>
<tr><td rowspan="8">0</td><td rowspan="4">00</td><td rowspan="2">000</td><td>0000</td></tr>
<tr><td>0001</td></tr>
<tr><td rowspan="2">001</td><td>0010</td></tr>
<tr><td>0011</td></tr>
<tr><td rowspan="4">01</td><td rowspan="2">010</td><td>0100</td></tr>
<tr><td>0101</td></tr>
<tr><td rowspan="2">011</td><td>0110</td></tr>
<tr><td>0111</td></tr>
<tr><td rowspan="8">1</td><td rowspan="4">10</td><td rowspan="2">100</td><td>1000</td></tr>
<tr><td>1001</td></tr>
<tr><td rowspan="2">101</td><td>1010</td></tr>
<tr><td>1011</td></tr>
<tr><td rowspan="4">11</td><td rowspan="2">110</td><td>1100</td></tr>
<tr><td>1101</td></tr>
<tr><td rowspan="2">111</td><td>1110</td></tr>
<tr><td>1111</td></tr>
</table>

Figure 5: Our symbol code supermarket where we can buy code words of length $l$ for a price of $2^{-l}$. Our budget is 1.

With this visual, there are two constraints that we can reintroduce. First is Kraft's inequality.

**Theorem 2.3 (Kraft Inequality)**

Every viable symbol code must have a budget $\leq 1$.

$$\sum_i 2^{-l_i} \leq 1 \tag{45}$$

If a symbol code achieves equality, then this is called a **complete symbol code**.

Second, a prefix code must have all codewords in different "rows" of the supermarket.

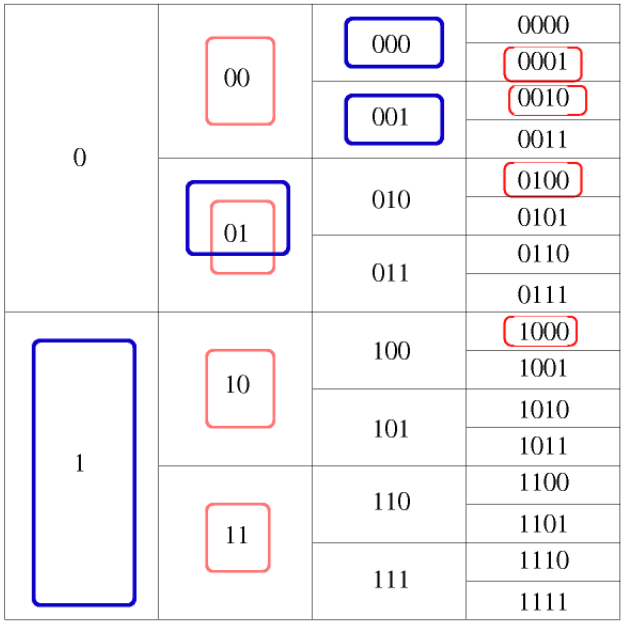



Figure 6: A symbol code that is a prefix code.

## 2.3 Huffman Coding

Now how well can we do with symbol codes? It turns out that the expected symbol code length cannot beat the entropy, and we will describe how to construct such a symbol code.

**Theorem 2.4 (Ideal Code Lengths)**

Given the expected length of a symbol code $C$ on an ensemble $X$, the expected length cannot be less than the entropy.

$$H(X) \leq \mathbb{E}[\ell(C(X))] = \sum_{s \in \mathcal{S}} \mathbb{P}(s) l_i \tag{46}$$

*Proof.* Such a code must exist. We first define the *ideal length* of the character $s_i$ with probability $p_i$ to be its surprisal

$$l_i^* = \log_2 \frac{1}{p_i} = \sigma_i \tag{47}$$

If you rearrange this and imagine someone that picked length $l_i$. We can pick an implicit probability $q_i$ satisfying the ideal length. It is $q_i = 2^{-l_i}$, but the person may not have chosen a complete code, so we must normalize it.

$$q_i = \frac{2^{-l_i}}{Z} \tag{48}$$

where $Z = 1$ if we have a complete code and $Z < 1$ if not. Therefore, we have

$$l_i = \log_2 \frac{1}{q_i} - \log_2 Z \tag{49}$$

We can then plug this into the expected length formula.

$$\begin{aligned}\mathbb{E}[\ell(C(X))] &= \sum_i p_i \left[ \log_2 \frac{1}{q_i} - \log_2 Z \right] \\ &= \sum_i p_i \log_2 \frac{1}{p_i} + \sum_i p_i \log_2 \frac{p_i}{q_i} - \log_2 Z \\ &= H(X) + D_{\text{KL}}(p \,||\, q) - \log Z \\ &\geq H(X)\end{aligned}$$

since the KL divergence (as we will show later) is greater than 0, and since $Z \leq 1$, we are subtracting a negative number.

To get equality, the proof shows you that you must make your implicit probabilities equal to your true probabilities, so the length of each character should be equal to its surprisal or information content. But these lengths aren't integers, so we must modify this in practice, which will get close, but not exactly to the true minimum. It turns out that we can get the expected length $L$ such that it is within 1 bit of the true minimum.

$$H(X) \leq L \leq H(X) + 1 \tag{50}$$

This is called the *Huffman algorithm*.

**Theorem 2.5 (Huffman Algorithm)**

The **Huffman algorithm** constructs an optimal prefix code for a given set of symbols and their probabilities. The algorithm proceeds as follows:

1. Create a leaf node for each symbol, and add it to a priority queue.
2. While there is more than one node in the queue:
   (a) Remove the two nodes with the lowest probability from the queue.
   (b) Create a new internal node with these two nodes as children, with a probability equal to the sum of their probabilities.
   (c) Add the new node back into the queue.
3. The remaining node is the root of the Huffman tree.
4. Traverse the tree, assigning 0 to each left branch and 1 to each right branch.
5. The Huffman code for each symbol is the sequence of 0s and 1s on the path from the root to that symbol's leaf node.

The Huffman algorithm produces an optimal prefix code, meaning that for any given set of symbols and probabilities, no other prefix code produces a smaller expected codeword length.

1. The expected length $L$ of the Huffman code satisfies:

$$H(X) \leq L < H(X) + 1 \tag{51}$$

2. The Huffman code is optimal among all prefix codes.
3. Symbols with higher probabilities get shorter codewords.
4. The algorithm has a time complexity of $O(n \log n)$ where $n$ is the number of symbols.

**Example 2.5 (Huffman Coding)**

Consider the alphabet $\mathcal{S} = \{A, B, C, D\}$ with probabilities $\{0.4, 0.3, 0.2, 0.1\}$.

1. Initial nodes: A(0.4), B(0.3), C(0.2), D(0.1)
2. Combine D and C: (D,C)(0.3), A(0.4), B(0.3)
3. Combine (D,C) and B: ((D,C),B)(0.6), A(0.4)

4. Final tree: (A,((D,C),B))(1.0)

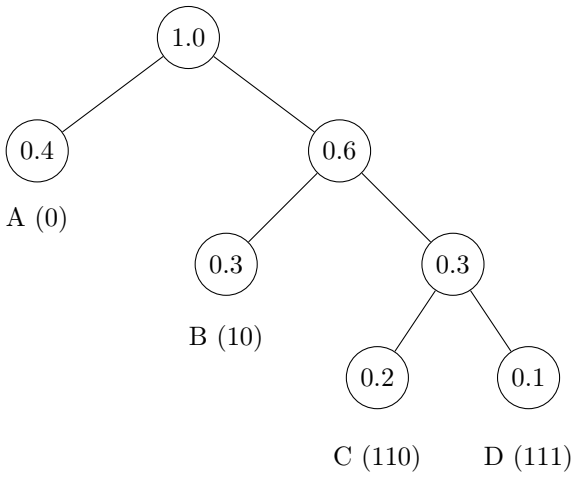



Figure 7: Huffman tree for the alphabet $\{A, B, C, D\}$

Resulting codes:
- A: 0
- B: 10
- C: 110
- D: 111

The expected length is $L = 0.4(1) + 0.3(2) + 0.2(3) + 0.1(3) = 1.9$ bits compared to the entropy of $H(X) = -\sum_i p_i \log_2 p_i \approx 1.85$ bits. This demonstrates that the Huffman code achieves a length very close to the entropy, and always within 1 bit of it:

$$H(X) \approx 1.85 \leq L = 1.9 < H(X) + 1 \approx 2.85 \tag{52}$$

## 2.4 Lempel-Ziv (LZ) Compression

The **LZ77** and **LZ78** (also known as the **LZ1** and **LZ2**), respectively, are lossless data compression algorithms published by Lempel and Ziv in 1977/78. They obsolete themselves but form the basis for many modern variations including **LZW, LZSS, LZMA**, and others.

LZ77 and 78 are both dictionary coders, but are not static. Rather, the dictionary starts in some predetermined state but the contents change during the encoding process.

## 2.5 Data Compression

**Data compression**, also called **source coding** or **bit-rate reduction**, is the process of encoding information using fewer bits than the original representation. Data compression algorithms can be categorzied into two types:

1. **Lossless compression** reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression.
2. **Lossy compression** reduces bits by removing unnecessary or less important information.

Typically, a device that performs data compression is referred to as an **encoder**, and one that performs the reversal of the process (decompression) as a **decoder**.

A **space–time** or **time–memory trade-off** in computer science is a case where an algorithm or program trades increased space usage with decreased time. Here, space refers to the data storage consumed in performing a given task (RAM, HDD, etc), and time refers to the time consumed in performing a given task (computation time or response time).

A space–time trade-off can be applied to the problem of data storage. If data is stored uncompressed, it takes more space but access takes less time than if the data were stored compressed (since compressing the data reduces the amount of space it takes, but it takes time to run the decompression algorithm). Depending on the particular instance of the problem, either way is practical. There are also rare instances where it is possible to directly work (which may also be faster) with compressed data.

## 2.6 Data Compression Ratio

The **data compression ratio**, also known as the **compression power**, is a measurement of the relative reduction in size of data representation produced by a data compression algorithm. It is defined as

$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}}$$

For example, a representation that compresses a file's storage size from 10MB to 2MB has a compression ratio of $10/2 = 5$. We can alternatively talk about the **space saving**, which is defined

$$\text{Space Saving} = 1 - \frac{\text{Compressed Size}}{\text{Uncompressed Size}}$$

So, the previous representation yields a space saving of 0.8, or 80%.

Lossless compression of digitized data such as video, digitized film, and audio preserves all the information, but it does not generally achieve compression ratio much better than 2:1 because of the **intrinsic entropy** of the data. Compression algorithms which provide higher ratios either incur very large overheads or work only for specific data sequences (e.g. compressing a file with mostly zeros). In contrast, lossy compression (e.g. JPEG for images, or MP3 and Opus for audio) can achieve much higher compression ratios at the cost of a decrease in quality, such as Bluetooth audio streaming, as visual or audio compression artifacts from loss of important information are introduced. In general, whether a compression ratio is high or not really depends on what kind of data is being compressed and how it is compressed.

## 2.7 Lossless Compression

Lossless data compression algorithms usually exploit statistical redundancy to represent data without losing any information, so that the process is reversible. Lossless compression is possible because most real-world data exhibits statistical redundancy. For example, an image may have areas of color that do not change over several pixels; instead of coding "red pixel, red pixel, ..." the data may be encoded as "279 red pixels". This is a basic example of run-length encoding; there are many schemes to reduce file size by eliminating redundancy.

A **dictionary coder**, also known as a **substitution coder**, is a class of lossless data compression algorithms which operate by searching for matches between the text to be compressed and a set of strings contained in a data structure (called the 'dictionary') maintained by the encoder. When the encoder finds such a match, it substitutes a reference to the string's position in the data structure.

Some dictionary coders use a *static dictionary*, one whose full set of strings is determined before coding begins and does not change during the coding process. This approach is most often used when the message or set of messages to be encoded is fixed and large; for instance, an application that stores the contents of a book in the limited storage space of a PDA generally builds a static dictionary from a concordance of the text and then uses that dictionary to compress the verses.

In a related and more general method, a dictionary is built from redundancy extracted from a data environment (various input streams) which dictionary is then used statically to compress a further input stream.

For example, a dictionary is built from old English texts then is used to compress a book. More common are methods where the dictionary starts in some predetermined state but the contents change during the encoding process, based on the data that has already been encoded.

### 2.7.1 Run-length Encoding (RLE) Compression

RLE is a form of lossless data compression in which *runs* of data (sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs.

For example, consider a screen containing plain black text on a solid white background. There will be many long runs of white pixels in the blank space, and many short runs of black pixels within the text. A hypothetical scan line, with B representing a black pixel and W representing white, might read as follows:

```
WWWWWWWWWWWWBWWWWWWWWWWWWWWBBBWWWWWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWWWW
```

With a RLE data compression algorithm applied to the above hypothetical scan line, it can be rendered as follows:

```
12W1B12W3B24W1B14W
```

which can be interpreted as a sequence of 12 Ws, 1 B, 12 Ws, 3 Bs, and so on. This run-length code represents the original 67 characters in only 18. While the actual format used for the storage of images is generally binary rather than ASCII characters like this, the principle remains the same. Even binary data files can be compressed with this method.

# References

[1] Claude Elwood Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.