What the bleat has texting got to do with thermodynamics?

A brief introduction to information theory

1 - Information

Consider a familiar situation - you and your friend want to talk about the adorable sheep that live in the fields between your houses. Tragically, however, it's getting late and they need to go back to their house. Not intending to let this end to discussion, you agree that you'll send messages by flashing lights out of your windows. Both of you have an LED light (which is coloured white), and a filament bulb (which is an orangey colour). But, converting messages into flashes of light and back again is pretty long winded and tedious, and the two of you want to find a way of encoding the messages that is as efficient as possible. But, how?

Imagine that, to start off with, you just want to tell your friend what noises the sheep is making. Let's consider two examples, one when the sheep is excited, and one when it's calm.

1) Excited: **BAABABABBAABBABABABAAAABABAA**
2) Calm: **BAAAAAAAAAAAAAAAABAAAAAAAAAA**

Importantly, when the sheep is excited and their baas are short and sharp, the message is split fairly evenly into Bs and As. But, when they're nice and calm, and the baas are longer, the message has far more As than Bs. In fact, when they're calm, the message is almost entirely just the letter A.

But, if you imagine actually sending those messages, doesn't the second one seem a bit silly? You're sending letter after letter, but the only time anything *interesting* happens is when a B appears. As an even clearer example, imagine if you sent a message where only 1 out of every 10,000 letters was a B - your friend would be bored out of their mind! In the most extreme situation, where every letter was guaranteed to be an A, then your message wouldn't be telling them *anything*.

The reason why I bring this up is to point out the fact that the amount of "content" each letter communicates intuitively seems to vary. But, does this actually mean anything in practise? Could we try and spend less effort sending letters which don't communicate much, or anything like that? Well, it seems like the answer is no. You've got two colours of light, so the best you can do is just pick one colour for each type of letter (A or B), but then you end up with one flash per letter in each case, so there's no difference! Admittedly, you could always try and send them using a different way of coding the messages all together, but for the sake of simplicity I'll avoid that. Anyway, as it turns out the reason we don't see a difference in this case is because we're using *discrete* data, where the values have to fit into distinct categories. This means that intermediate values have to be rounded, so previously different

numbers can end up looking the same. Fortunately, this is easy to deal with in situations like this, by simply adding more possible values of data. That way, close numbers are less likely to be rounded to the same value.

Let's look at an extreme example. Instead of sending messages made up mostly of Bs and As, this time you tell your friend you'll use different letters to communicate subtle differences in the baas. We'll again consider two cases, one where the sheep are all starting and stopping a lot, and one where they make very, very long baas (only 1 in every couple hundred characters isn't an A).

1) Rapid baas: **FCSAITOPABJVHTPBVAZTZASJBAAW**
2) Long baas: **NAAAAAAAAAAAAAAAAAAAAA**

In case one, you can't distinguish between the letters, so you might just give them codes of the same length. In this case, they'd have to be five flashes each (each flash doubles the number of possible characters you can use, and the smallest power of 2 greater than 26 is $32 = 2^5$). But, in the second case, if you let A be just a green flash, and the rest of the characters a 5 flash code starting with a blue flash, then on average you only need a little over 1 flash per character. That's a huge difference! So, we've (sort of) confirmed that this "content" idea might actually mean something practical. The next thing to do then is to make it more rigorous.

Firstly, we need to define a "source" of information, which we'll always represent with the letter X. A source is anything that, when checked, randomly returns one of a discrete set of "outputs". These possible outputs will be written as $x_1, x_2, ..., x_i, ..., x_n$. Importantly, the chance of getting any particular output has to be the same every time the source is checked, and the probability of getting some output x_i will be written as p_i. Lastly, the number of outputs for a given source will usually be called n. In the earlier example, the source was the sheep, and the outputs were the letters in your message. Obviously in real life the chance a sheep makes any given noise is not a constant value at every point in time, but you can roughly approximate it like that.

Then, we need to define our "content" or "information" property for the outputs of sources. Let's list some properties we'd intuitively like it to have.

1) It makes sense that the information you get from an output of a source doesn't specifically depend on what the source or output is. That is, it doesn't matter if it's a sheep making noise or something else. Instead, we just want the information to be based on the *probability* of the output. So, we'll say that our "information" idea is a *function* that takes a probability and returns some number. More formally, we'll define it as $I : [0, 1] \to \mathbb{R}$. That is, it's the function I that maps numbers in the range 0 to 1, to any other set of points on the number line.

2) Next, it makes sense that if an output is guaranteed to happen, you don't learn anything from finding out it did. Again, we can write this as $I(1) = 0$.

3) More generally, the less likely an output is, the more interesting it is when it actually happens, and the more common it is, the less interesting it is to find out that it happened. This means that $I(p)$ is *monotonically decreasing*. As an equation, we can write this as $I(p+\epsilon) < I(p), \quad 0 < \epsilon < 1 - p$.

4) Lastly, we want to be able to combine the information from multiple outputs. So, why don't we say that the information you get from output A and B from happening, is just the sum of the information you get if they happened individually, so we know that $I(p_{AB}) = I(p_A) + I(p_{B|A})$. But, we know that $p_{AB} = p_A \cdot p_{B|A}$, so you can write the rule simpler by saying that $I(p_1 \cdot p_2) = I(p_1) + I(p_2)$.

Surprisingly with just those four constraints, you can actually show that $I(p)$ *has* to be one of a small set of functions, each of which is just a scalar multiple of all the others!

Firstly, we can take derivative of the equation in rule 4 in respect to one of the two probabilities.

$$p_1 \cdot I'(p_1 \cdot p_2) = I'(p_2)$$

Then, set $p_2 = 1$.

$$p_1 \cdot I'(p_1) = I'(1)$$

Divide by p_1

$$I'(p) = I'(1) \cdot \frac{1}{p}$$

Then, by the fundamental theorem of calculus,

$$I(p) = I'(1) \int_1^p \frac{1}{t} dt + I(1)$$

Which is equal to,

$$I(p) = k \ln(p)$$

To ensure it follows rule 3, k must be negative, so we can write it in the form

$$I(p) = -log_a(p) \quad , a > 1$$

As a note, different values of a are commonly used, such as 2, e, and 10. For the sake of convenience, we'll be exclusively using the natural log, but it can be easily converted to any other base by dividing by $ln(a)$

From this, we can also define the *average* information you'd expect per output for some source, which we'll write as $H(X)$. To do this, you just multiply the likelihood of a given output by the amount of information you get from it, and just add it all together.

$$H(X) = \sum_{i=1}^{n} p_i \cdot I(p_i) = -\sum_{i=1}^{n} p_i \ln(p_i)$$

2 - Definitions

Great! We've taken this confusing, poorly defined idea of "information" and turned it into a well defined (albeit still confusing) mathematical function. But, how is this going to help you send messages efficiently? Well, like always, to answer this we have to make the situation more rigorous.

Firstly, what *exactly* does encoding mean? This section is going to involve a lot of definitions, so sorry if it's a little difficult to keep track of. Let's suppose you have an information source X, with a set of possible outcomes $S = x_1, x_2, ...x_n$, and a set of matching probabilities $P = p_1, p_2, ..., p_n$ describe how likely each outcome is.

Then, you need an alphabet of characters T, of length r, with which the outcomes will be encoded. The set of all possible "words" (which we'll refer to as strings) you can make with this alphabet is written as $T*$. A "code" is then simply any function F that maps each outcome in S to some string in $T*$. That is, a code is any function $F : S \to T*$. The subset of $T*$ that is mapped to by F (often called the *image* of S under F) will be called C. Lastly, for every outcome x_i,we'll write the length of the string that it's encoded by as ℓ_i.

Simple, right?

However, we're not just dealing with *any* possible code. Instead, we're only interested in *prefix* codes, where none of the strings in C are the beginning of any others (personally, I think prefix-*less* makes more sense). This means that the code in question is *uniquely decodable*; since you can tell unambiguously where characters start and end, you can always work out the original series of outputs. This limitation is largely just for convenience, but the actual results apply to any uniquely decodable code (and besides, common examples like ASCII or Unicode are prefix codes).

The only thing left is to be more specific about efficiency. In this case, we'll say that the efficiency of a code depends on the expected length of string you use for each outcome. The most efficient code will be any one that minimises the expected length (although there's not guaranteed to only be one). The formula for the expected length is the length of each string multiplied by how likely it occurs, or:

.

$$E(\ell) = \sum_{i=1}^{n} p_i \cdot \ell_i$$

Now we can finally find the most efficient code for any given X and T!

Except, we won't. As it turns out, this is surprisingly simple, and if you've followed the essay up to this point you can probably learn it in just a few minutes (known as Huffman Coding). So, why on Earth would I pose that as the key question if I had no intention of answering it? Basically, because it's easy to ask. We've just gone through multiple paragraphs of definitions and clarifications about what situation we're even dealing with; can you imagine if all of this was *before* the introduction?

The actual key question is: what is the lower bound on the efficiency of a prefix code?

The reason why this is interesting is because it turns out the average information function we defined earlier, $H(X)$, is exactly that! Personally, I think that this is really interesting. An intuitive way of thinking about it is that if the information from the outputs was greater than the information of the code, then you'd have to have lost some of it, and wouldn't be able to work out the original series of outcomes.

In order to prove this, there are unfortunately two other results we need to use. For space reasons, I can't include full proofs.

3 - Kraft-McMillan and Gibbs' inequalities

The Kraft-McMillan inequality relates the minimum length of strings used in a uniquely decodable code to the number of outcomes that need to be encoded, and the number of characters available. The specific version we need can be stated as:

For any prefix code, $$\sum_{i=1}^{n} r^{-\ell_i} \leq 1$$

To show this, I'm going to use a type of diagram called a "tree", and some common terminology, which I'd recommend looking up if you're not familiar with it already.

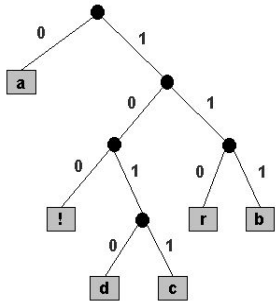

figure 1

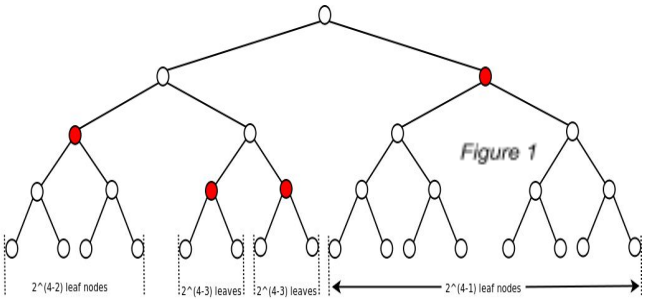

figure 2

We can represent any prefix code as an r-ary tree. The strings representing each outcome can be found by reading down along the branches from the top node to the outcome. Figure 1 is an example of this, when $r = 2$.

We can complete the tree, up to some layer L, as in figure 2 (again, letting $r = 2$). The bottom layer has r^L nodes in it. Similarly, each coding node has $r^{L-\ell_i}$ daughter nodes in the bottom layer. In a prefix code, no coding node is a daughter node to any other coding node, so no two coding nodes share any daughter nodes in the bottom layer either. As a result,

$$\sum_{i=1}^{n} r^{L-\ell_i} \leq r^L$$

By dividing each side by r^L, we get the inequality.

Secondly, there's the Gibbs' inequality. Informally, this states that the average information function we defined earlier is minimised when the probabilities used to find the information associated with each outcome are the same as the actual probabilities of each outcome. More formally, for any two probability distributions P and with $Q$the same number of possible outcomes,

$$-\sum_{i=1}^{n} p_i \ln(p_i) \leq -\sum_{i=1}^{n} p_i \ln(q_i)$$

With a little algebra, this can be rearranged as,

$$-\sum_{i=1}^{n} p_i \ln(\frac{q_i}{p_i}) \geq 0$$

which can be shown by using the fact that $\ln(x) \leq x - 1$.

4 - Shannon's Source Coding theorem

For a given uniquely decodable code, we'll define a probability distribution Q such that

$$q_i = \frac{r^{-\ell_i}}{\sum_{j=1}^{n} r^{-\ell_j}}$$

Then, starting with the definition of H:

$$H(X) = -\sum_{i=1}^{n} p_i \ln(p_i)$$

Gibbs' inequality

$$\leq -\sum_{i=1}^{n} p_i \ln(q_i)$$

$$= \sum_{i=1}^{n} p_i(-\ell_i)\ln(r) + \sum_{i=1}^{n} p_i \ln(\sum_{j=1}^{n} r^{-\ell_j})$$

$$= \ln(r)\sum_{i=1}^{n} p_i \cdot \ell_i + \ln(\sum_{j=1}^{n} r^{-\ell_j})$$

Kraft-McMillan inequality

$$\leq \ln(r) \sum_{i=1}^{n} p_i \cdot \ell_i$$

$$= \ln(r) \cdot E(\ell)$$

$$E(\ell) \geq H(X)/ln(r)$$

And, Information Theory was discovered.

Personally, I think that this is *stunning*. There are actual, tangible results you can show about communication and information, despite how crazy and messy it is. Of course, this relies on a ridiculously simplified model of information and communication, but even in this case it's remarkable that these results can exist.

But, when it comes down to it the way I feel about certain pieces of maths doesn't mean much for its significance in the wider world. So, why does anyone care about this? Well, modern communications technology is built from Information theory. Computers, the internet, data analysis, machine learning, cryptography, and so on, all depend on it. Fields of study like linguistics, cognitive science, and quantum physics have picked up its key ideas. In fact, the word bit was first used in a publicly published paper as the unit for H, and now it's one of the most common words that came from computer science.

And so, we can move onto the last topic, and go over a sneaky piece of information I've left out so far.

5 - Thermodynamics

In physics, you can generally classify descriptions of a physical system (some collection of physical things) into one of two categories: ones that relate to the system on a large, or "regular" scale, and ones that describe it on smaller and more precise scales. For example, the size and shape of an object is one of these larger scale properties, while the specific arrangements of the molecules inside it is a smaller scale one. The properties of physical systems on these different scales are usually referred to as macrostates (from the Ancient Greek for long, or large), and microstates (from the Ancient Greek for small).

Importantly, for any given macrostate of a system (what people usually consider as its properties), there are many possible microstates. As an example, imagine someone connects two balloons of air by a small pipe. If each balloon has about 1 mole of gas molecules in it, then there are over $10^{10^{20}}$ different ways of allocating those molecules between the two halves of the system. That number is, frankly, incredibly big.

Before we start doing maths again, there's one last observation to make about these microstates. Given a physical system, we can act like every microstate is equally likely to

occur. Practically this might not be true, but at least given the kind of model we're making it would be very difficult to distinguish between them, and as it turns out you can make very accurate predictions about thermodynamics by assuming that. We're also going to assume that microstates are mutually exclusive, so a system can't be in multiple ones at once.

So, back to maths. Given a system with N possible microstates, and a set of possible microstates of size n, the chance the system is in a state in that set is given by n/N. If that set describes a possible macrostate, this tells us that the chance that a system is in a particular macrostate is proportional to the number of possible microstates that fit the description of that macrostate. From this, we can conclude that the most likely macrostate for a system to be in is the one with the most corresponding microstates.

But, wait a minute. This whole description of systems seems very similar to the information sources we were talking about before. There's something you can check (a system in a particular macrostate), and when you do you can observe one of some number of possible outcomes (the microstate), and there's a corresponding probability for each outcome (in this case, $1/N$). So, that means we can calculate an H value for it!

$$H(X) = -\sum_{X} p_i \ln(p_i) = -\sum_{X} \frac{1}{N} \ln(\frac{1}{N}) = -N \cdot \frac{1}{N} \cdot -\ln(N) = \ln(N)$$

Which is monotonically increasing with regards to N. Combining these two results tells us that the most likely macrostate to observe a system in is the macrostate with the highest H value, and presumably then if a system can change over time to a state with a higher H value, that would be the statistically likely option. With smaller systems, it's reasonable that it might not end up in that state, but as the size increases the chance that it's found in that state scales very quickly. Going back to the balloon example, the probability that 50% of the molecules are in each balloon, is at least $10^{10^{22}}$ times higher than the chance that 49% are in one and 51% are in the other. Because of this, if a large system can change in a way that increases H, it seems almost guaranteed that it would try and move in that direction, to the point that someone so inclined might even call it a physical law.

What's the sneaky piece of information I've left out?

The name of H *is information entropy*.

And, likewise, that mess is the second law of thermodynamics. Entropy has a reputation for being one of the most confusing and difficult concepts in physics, but hopefully this essay's managed to give you some insight into it that you might not have had before. Of course, the concept of entropy in physics isn't *actually* the same, and in the wider context of physics it's treated entirely differently than in Information theory. But, they're still incredibly closely linked concepts. Personally, I think that introducing entropy from the (relatively) more intuitive concept of information entropy would help quite a lot of people trying to get their head around it. That's why I ended up writing this essay in the first place.

So, to come back to the title, what does texting have to do with thermodynamics?

Apparently, quite a lot.

Lastly, I thought it might be interesting to mention why Claude Shannon, the mathematician who introduced all of these ideas, used the phrase information entropy in the first place, as the story very accurately describes both my own experience looking at entropy, but also that of many, many other maths and physics students who have unfortunately stumbled upon it. Supposedly John von Neumann (another incredibly influential mathematician and physicist) insisted that Shannon should, saying:

“You should call it entropy, for two reasons. In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more importantly, no one really knows what entropy really is, so in a debate you will always have the advantage.”