The random coding argument: digital communication

In terms of math, this note is about a powerful and surprising use of probability — it can be used to prove the existence of useful objects even when we might not be able to give explicit constructions for those objects.

At the practical level, this note is motivated by the desire for error correcting codes that work at the bit level rather than at the packet level. The core arguments date back to Claude Shannon's revolutionary 1948 paper entitled "A Mathematical Theory of Communication." Shannon was truly one of the great minds of the 20th century and is one of those responsible for bringing forth the digital age. His 1937 master's thesis is credited with connecting the then seemingly esoteric subject of Boolean Algebra to the design of digital circuits and switching systems, and showing that such circuit design could be done in a systematic way. His 1940 PhD dissertation brought mathematics to the theory of genetics, and then he found himself working on both fire-control systems as well as secrecy/cryptography during World War II. Shannon established the fundamental limits on secrecy[1], but to do so, he had to better understand what was the essential heart of communication itself. His 1948 paper firmly established the fundamental nature of "bits" and what reliable communication can/should mean and seeded the modern field of Information Theory.

The Wikipedia article on Claude Shannon is pretty good and quite inspiring. You are encouraged to read it.

Communication over a Noisy Channel

The fundamental problem in both data-links and data-storage is the same — communication over a noisy channel. In a link, the communication usually spans space: some rapidly moving information-carrying medium (light, radio waves, sound waves, electrons in a wire, etc.) is modulated at the sender to carry the message and is then received at the destination a very short time later where it is captured by some detector. In storage, the communication spans time: some stable information-carrying medium (pigments, magnetic domains, a capacitor, etc.) are configured by the recorder to hold the message. This medium is later read by some detector. In both cases, the original message must be reliably extracted from the output of the detector. The challenge is that all known physical mechanisms for carrying information are subject to noise — chance fluctuations that can corrupt what was originally sent. For example, the random motion of electrons due to finite non-zero temperatures, the cosmic background radiation left over from the Big Bang, the actions of cosmic rays and natural ambient radioactivity, as well as interfering signals — all of these are often modeled as noise since they are not maliciously trying to deceive us, but impair our ability to communicate nonetheless.

One of the simplest models is called the binary symmetric channel. Let X_i be the i-th input bit to the channel — what was sent by radio or written to disk. This is something that the message encoder gets to choose based on the message, but is restricted to be either 0 or 1. The i-th output of the channel is denoted Y_i and this equals X_i with probability $1-p$ and is flipped with probability p. Mathematically, we can view X and Y as taking values in the binary field GF(2) and let Z_i be an i.i.d. sequence of Bernoulli random variables with parameter p. ($Z_i = 1$ with probability p and $Z_i = 0$ with probability $1-p$.) Then $Y_i = X_i + Z_i \bmod 2$. Note

[1]The paper was entitled "A Mathematical Theory of Secrecy Systems" and was promptly classified. It was declassified in 1949.

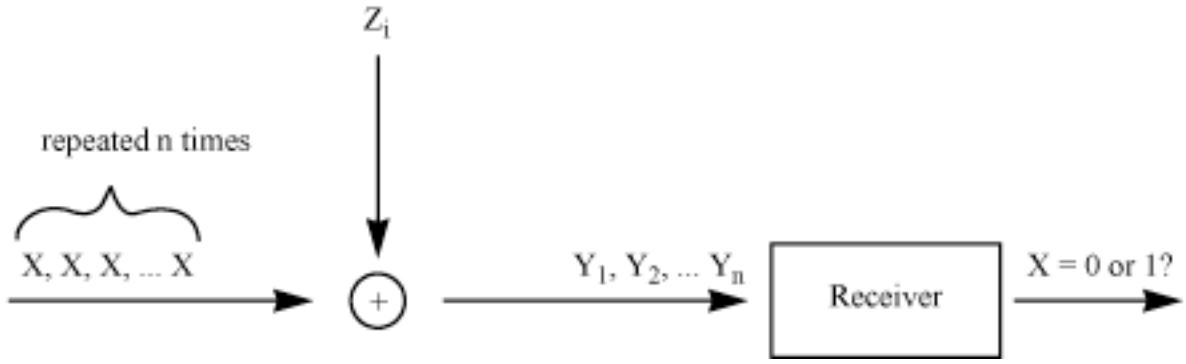

Figure 1: A repetition code

that Y_i is different from X if and only if $Z_i = 1$. Reflecting the non-maliciousness of the channel, the Z_i are independent of the message and the encoding scheme.

If you want to make everything concrete in your mind, think of $p = 0.05$ which reflects a situation in which on average, one in twenty bits get flipped. Is there any hope of achieving reliable transmission or reliable storage of information in the face of such corruption?

The repetition code

The easiest way to increase the reliability of a system is to add redundancy — just repeat the desired bit over and over again. The situation is shown in Figure 1.

Question: I have one bit of information that I want to communicate over a noisy channel. The noisy channel flips each one of my transmitted symbols independently with probability $p < 0.5$. How much improvement in performance do I get by repeating my transmission n times?

Comment: In an earlier lecture note, we also considered a communication problem and gave some examples of error-correcting codes. However, the models for the communication channel are different. There, we put a bound on the maximum number of flips the channel can make. Here, we do not put such bounds *a priori* but instead impose a bound on the *probability* that each bit is flipped (so that the *expected* number of bits flipped is np). Since there is no bound on the maximum number of flips the channel can make, there is no guarantee that the receiver will always decode correctly. Instead, one has to be satisfied with being able to decode correctly *with high probability*, e.g., probability of error < 0.001.

In the repetition code, each $X_i = X$. Notice that the received symbols Y_i's are *not* independent; they all contain information about the transmitted bit X. However, *given* X, they are (conditionally) independent since they then only depend on the noise Z_i.

Decision rule

First, we have to figure out what *decision rule* to use at the receiver, i.e., given each of the 2^n possible received sequences, $Y_1 = b_1, Y_2 = b_2, \ldots Y_n = b_n$, how should the receiver guess what value of X was transmitted?

For this particular example, there is an obvious choice: a *majority* rule: guess that a 0 was transmitted if the number of 0's in the received sequence is at least as large as the number of 1's, otherwise guess that a 1 was transmitted.

Why is this a good choice? The intuition is pretty simple. Look at the Zs. As long as $p < 0.5$, 0s are more likely than 1s. So, it makes sense to bet that fewer raw bits were flipped rather than more raw bits!

(*You can skip the rest of this section if you want.)*

How would we come to this obvious choice in a more mechanical way? A natural rule is the *maximum a posteriori* (MAP) rule: guess the value a^* for which the conditional probability of $X = a^*$ given the observations is the largest among all a. More explicitly:

$$a^* = \begin{cases} 0 & \text{if } \Pr[X=0|Y_1=b_1,\ldots,Y_n=b_n] \geq \Pr[X=1|Y_1=b_1,\ldots Y_n=b_n] \\ 1 & \text{otherwise} \end{cases}$$

Now, let's reformulate this rule so that it looks cleaner. By Bayes' rule, we have

$$\Pr[X=0|Y_1=b_1,\ldots Y_n=b_n] = \frac{\Pr[X=0]\Pr[Y_1=b_1,\ldots,Y_n=b_n|X=0]}{\Pr[Y_1=b_1,\ldots,Y_n=b_n]} \quad (1)$$

$$= \frac{\Pr[X=0]\Pr[Y_1=b_1|X=0]\Pr[Y_2=b_2|X=0]\ldots\Pr[Y_n=b_n|X=0]}{\Pr[Y_1=b_1,\ldots,Y_n=b_n]} \quad (2)$$

In the second step, we are using the fact that the observations Y_i's are conditionally independent given X. (Why?) Similarly,

$$\Pr[X=1|Y_1=b_1,\ldots Y_n=b_n] = \frac{\Pr[X=1]\Pr[Y_1=b_1,\ldots,Y_n=b_n|X=1]}{\Pr[Y_1=b_1,\ldots,Y_n=b_n]} \quad (3)$$

$$= \frac{\Pr[X=1]\Pr[Y_1=b_1|X=1]\Pr[Y_2=b_2|X=1]\ldots\Pr[Y_n=b_n|X=1]}{\Pr[Y_1=b_1,\ldots,Y_n=b_n]} \quad (4)$$

An equivalent way of describing the MAP rule is that it computes the ratio of these conditional probabilities and checks if it is greater than or less than 1. If it is greater than (or equal to) 1, then guess that a 0 was transmitted; otherwise guess that a 1 was transmitted. (This ratio indicates how likely a 0 is compared to a 1, and is called the *likelihood ratio*.) Dividing (2) and (4), and recalling that we are assuming $\Pr[X=1] = \Pr[X=0]$, the likelihood ratio L is:

$$L = \prod_{i=1}^{n} \frac{\Pr[Y_i=b_i|X=0]}{\Pr[Y_i=b_i|X=1]}. \quad (5)$$

Note that we didn't have to compute $\Pr[Y_1=b_1,\ldots,Y_n=b_n]$, since it appears in both of the conditional probabilities and got canceled out when computing the ratio.

Now,

$$\frac{\Pr[Y_i=b_i|X=0]}{\Pr[Y_i=b_i|X=1]} = \begin{cases} \frac{p}{1-p} & \text{if } b_i=1 \\ \frac{1-p}{p} & \text{if } b_i=0 \end{cases}$$

In other words, L has a factor of $p/(1-p) < 1$ for every 1 received and a factor of $(1-p)/p > 1$ for every 0 received. So the likelihood ratio L is greater than 1 if and only if the number of 0's is greater than the number of 1's.

Thus, the decision rule is simply a *majority* rule: guess that a 0 was transmitted if the number of 0's in the received sequence is at least as large as the number of 1's, otherwise guess that a 1 was transmitted.

Note that in deriving this rule, we assumed that $\Pr[X=0] = \Pr[X=1] = 0.5$. When the prior distribution is not uniform, the MAP rule is no longer a simple majority rule.

Error probability analysis

What is the probability that the majority-rule guess is incorrect? This is just the event E that the number of flips by the noisy channel is greater than $n/2$. So the error probability of our majority rule is:

$$\Pr[E] = \Pr\Big[\sum_{i=1}^{n} Z_i > \frac{n}{2}\Big] = \sum_{k=\lceil n/2 \rceil}^{n} \binom{n}{k} p^k (1-p)^{n-k},$$

recognizing that the random variable $S := \sum_{i=1}^{n} Z_i$ has a binomial distribution with parameters n and p.

This gives an expression for the error probability that can be numerically evaluated for given values of n. Given a target error probability P_e of, say, 0.001, one can then compute the smallest number of repetitions needed to achieve the target error probability.[2]

As in the hashing application we looked at earlier in the course, we are interested in a more explicit relationship between n and the error probability to get a better intuition of the problem. The above expression is too cumbersome for this purpose. Instead, notice that $n/2$ is greater than the mean np of S and hence the error event is related to the tail of the distribution of S.

We now have multiple possibilities for what to do.

- One choice is to apply Chebyshev's inequality to bound the error probability:

  $$\Pr\Big[S > \frac{n}{2}\Big] < \Pr\Big[|S-np| > n(\frac{1}{2}-p)\Big] \leq \frac{\mathrm{Var}(S)}{n^2(\frac{1}{2}-p)^2} = \frac{p(1-p)}{(\frac{1}{2}-p)^2} \cdot \frac{1}{n},$$

  using the fact that $\mathrm{Var}(S) = n\mathrm{Var}(Z_i) = np(1-p)$.

  The important thing to note is that the error probability decreases with n, so indeed by repeating more times the performance improves (as one would expect!). For a given target error probability of, say, 0.001, one needs to repeat no more than

  $$n = 1000 \cdot \frac{p(1-p)}{(\frac{1}{2}-p)^2}$$

  times. For $p = 0.05$, this evaluates to 235.

  This is a pretty pessimistic result. It says that to bring the probability of error down from 0.05 to 0.001 — a factor of 50 reduction — we need around 235 repetitions. And this would seem to be growing as $O(\frac{1}{P_e})$ if $P_e \to 0$. Are things really that bad?

- A faster cheaper hack is to simply apply the Central Limit Theorem. We expect np flips. We're in trouble if we reach $\frac{n}{2}$ flips or more. According to the CLT, the question boils down to how many standard deviations away that is. Looking at a Gaussian table says that the tail probability beyond 3.1 standard deviations on one side is about 0.001. In other words $Q^{-1}(0.001) \approx 3.1$ where $Q(t) = \int_t^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$ for $t > 0$.

  For a given n, the standard deviation is $\sqrt{p(1-p)n}$ and so we want $pn + 3.1\sqrt{p(1-p)n} = \frac{n}{2}$, which is the same as saying $(\frac{1}{2}-p)\sqrt{n} = 3.1\sqrt{p(1-p)}$ or $n = (3.1)^2 \frac{p(1-p)}{(\frac{1}{2}-p)^2}$. For $p = 0.05$, this gives us an $n \approx 2.25$ or practically speaking, $n = 3$.

[2] Needless to say, one does not want to repeat more times than is necessary as we are using more time or space to communicate each information bit and the rate of communication is reduced. We like fast transmission of information and high information densities for storage, if we can get them reliably.

This just feels wrong. Not only is it a lot smaller than 235, but if $n = 3$, then two or more flips will cause an error. But the probability of 2 flips is already 0.007125 and 3 flips is another 0.000125, for a total probability of error of 0.00725, which is certainly greater than 0.001.

So, the CLT approximation was too aggressive[3] in this case. But are we even close? What if we go to the next reasonable value of $n = 5$. Then, three or more flips cause an error. Three flips have a probability of about 0.00113 with four flips being much less likely. So 5 is almost right.

$n = 7$ will work since we the probability of getting 4 flips is a mere 0.00019 or so, and beyond 4 flips is negligible by comparison.

So, we find ourselves in a situation where Chebyshev's inequality is far too conservative while the CLT is aggressive, but is kind-of in the right ballpark. Clearly we need a better way to proceed.

- The better way to proceed is represented by Chernoff's bound. For the case of Bernoulli-p random variables, Chernoff's bound[4] says that for $a > p$, we know $P(\sum_{i=1}^{n} Z_i \geq an) \leq e^{-nD(a||p)}$ where $D(a||p) = a\ln\frac{a}{p} + (1-a)\ln\frac{1-a}{1-p}$. This particular D function is called the binary *divergence*.

  Evaluating $D(0.5||0.05)$ in our case gives 0.83. We want to choose an n so that $e^{-nD(0.5||0.05)} = 0.001$, and so this bound says that $n = 9$ is certainly safe.

  Chernoff's bound is a bit conservative, but not by a lot. The bound wanted us to use something a little bigger than 8. Meanwhile 7 was safe.

So a repetition code lets us get to a rate of 1 data bit for every 7 raw bits or a rate of about 0.143 bits/raw-bit. This is for a target error probability of 10^{-3} and a raw error rate of 0.05.

Furthermore, both Chernoff, Chebyshev, and the CLT agree that if we were to demand increasing reliability and lower target error probabilities, this rate would get ever lower. Chernoff says that the rate would drop like $O(\frac{1}{\ln(\frac{1}{P_e})})$ as $P_e \to 0$.

Is this good or bad? Can we even hope for anything better? Can we do any better?

Trying to use Reed-Solomon codes

You are encouraged to take what you've learned above and apply it to using Reed-Solomon codes for this problem. The key challenge is that Reed-Solomon codes work at the packet level rather than the individual bit level. This is because the codewords are fundamentally the evaluations of an underlying polynomial at different points. A single bit by itself comes from GF(2) and there are only two points at which to evaluate anything! Clearly, the laws of large numbers cannot help us with only two points. To get more points, we need bigger packets.

The trick is to create convoys of bits that travel together. ℓ bits can be grouped together into an ℓ bit packet — and fortunately it turns out that there exist[5] finite fields GF(2^ℓ). The problem is that when we do this, the

[3] What went wrong? What is wrong is that the gap between p and $\frac{1}{2}$ is too big. The Central Limit Theorem has the word "Central" in its name for a reason! It is meant to be a good approximation to small deviations from the average.

[4] There are many ways of seeing this, and the HW asks you to derive a close relative of this particular bound. Can you see why Stirling's approximation would also hint at this particular form? Hint: look at Stirling's approximation as applied to the dominant binomial coefficient.

The other straightforward way to derive this is to take the event $\{\sum_{i=1}^{n} Z_i \geq an\}$ and use the monotonicity of a growing exponential to say that this is just the same event as $\{e^{r\sum_{i=1}^{n} Z_i} \geq e^{ran}\}$ as long as $r > 0$. Then apply Markov's inequality, use the fact that the exponent of a sum is the product of exponents, use the fact that the expectation of a product is the product of expectations for independent random variables, and then leverage the identically-distributed nature of things to pull all the n terms together with algebra. At this point, just optimize over the choice of r and you get this bound after some basic calculus and algebra.

[5] More on these in Math 113 and 114. They come from viewing polynomials as numbers of a sort.

raw probability of error on these packets is larger than it was for the original bits. For example, for $\ell = 8$, the raw probability of error at the byte level jumps to 0.3366 or so because the byte will be corrupted if any bit within it is corrupted.

And how shall we deal with these corrupted packets? We can try to use the Berlekamp-Welch decoding algorithm from the previous notes to solve this. After all, that algorithm (which maps the problem into solving a linear system of equations) is guaranteed to be able to correct up to a certain number of malicious errors so it will certainly work on that same number of random errors.

It is an excellent and highly recommended exercise for you to actually do this analysis. It turns out for $\ell = 8$ and $p = 0.05$, that about we can use a code with 256 points in it. Of these, even with a 3.1 sigma error event, no more than 110 of these bytes will be corrupted. That means that 256-2*110 = 36 bytes of payload data can be reliably carried by the code. This gives a rate of 36 data bytes per 256 raw bytes or 0.14.

It turns out that as we vary ℓ to be smaller than 8, the net rates are lower because we have shorter codes and consequently must budget for proportionately larger random fluctuations. When ℓ is larger than 8, the probability of corruption of at the packet level gets higher and although the random fluctuations that must be budgeted for are smaller, the average is bigger. Once again, the rates become lower. Once ℓ exceeds 13, this approach just stops working because more than half the packets have errors on average.

Furthermore, this approach breaks down if we try to approach probabilities of error that are very small. It simply will not let us reach probabilities of error around 10^{-11} or lower. Meanwhile, the Chernoff bound tells us that using a repetition code of length 31 would certainly give us that kind of reliability.

Random Coding

Repetition codes keep bits isolated but have rates that tend to zero as reliability demands increase. The Reed-Solomon codes let bits convoy together to share redundancy among them, but turn out not to work well if we use the Berlekamp-Welch decoding approach.

So, is there any hope of doing better?

The first part of Claude Shannon's genius was to actually ask the question in such a way that there was some hope to answer it. We need to make the idea of encoding and decoding a little bit more precise.

We have m distinct potential messages. We would like to encode them using n raw bits. We want to do so in such a way that when these encoded bits get corrupted by random noise with a probability p of flipping a bit, that the decoder can reliably infer the message that was actually sent.

- The set of m messages represents m potential things that the encoder wants to encode. Without loss of generality, we consider these to simply be the m natural numbers $0, 1, \ldots, m-1$. The numbers might mean an answer to the question "How much wood could a woodchuck chuck if a woodchuck could chuck wood?" Or the messages might simply represent a list of strings so 0 = "Cake is so delicious and moist," 1 = "The cake is a lie," 3 = "I'm a potato," 4 = "Make life take the lemons back," 5 = "Mantis men," 6 = "I'm in space." But it doesn't matter. The meaning is irrelevant for communication. What matters is that both the encoder and the decoder agree that exactly one of these will be transmitted, but the decoder doesn't know which one.

- The *codebook* is a mapping that takes the messages into *codewords.* Each codeword is an n-length binary string. The bits in the codewords are called codebits. Both the encoder and the decoder know the entire codebook. The codeword for message s is denoted $x_1(s), x_2(s), x_3(s), x_4(s), \ldots, x_n(s)$.

- The true message S is the message that we want to convey. The corresponding codeword is the *true*

codeword $X_1, X_2, \ldots X_n$ and is actually transmitted. The encoder knows the true message and hence the true codeword. The decoder doesn't know the true message and must try to infer it by trying to detect which codeword was sent by using its received string $Y_1, Y_2, \ldots Y_n$. The decoder's guess for the true message is called the *decoded message* and is represented by $\hat{S}$.

- All codewords other than the true codeword are called the *false codewords*.
- We say that the decoder has made an error if the decoded message does not equal the true message.
- The rate of the code measured in data bits per raw bit is defined to be $\frac{\log_2 m}{n}$ because we need $\log_2 m$ data bits to uniquely specify which message is the true one. Since correct decoding lets the decoder know which message is the true one, it can be considered as successfully communicating $\log_2 m$ bits.

The goal is to have the probability of error be small for every potential message in the codebook.

Too many codebooks

So why don't we simply search over the set of all codebooks? Let's count them first. Given the messages, each codebook is uniquely defined by a list of m codewords which is essentially an mn-length bitstring all together. Consequently there are 2^{mn} different codebooks of length n for m messages.

To see why this is way too big, let's assume that we want to have the rate of the code be $\frac{1}{2}$ data bits per raw bit. Then, we have the following table:

message length	m	n	2^{nm}
1	2	2	16
2	4	4	$2^{16} = 65536$
3	8	6	$2^{48} = 281$ Trillion
4	16	8	$2^{128} = 3.4 \times 10^{38}$
5	32	10	$2^{320} = 2.1 \times 10^{96}$

For comparison, it is good to remember that it has only been 4.4×10^{23} microseconds since the Big Bang, the mass of the Solar system is only about 2×10^{36} milligrams and the number of atoms in the observable universe is estimated to be about 10^{80}. So these are absurdly large numbers of codebooks.

At first glance, asking the question this way seems hopeless. It feels like asking whether there is a needle hidden somewhere in a universe-sized haystack.

How to decode in principle

Before we describe Shannon's brilliant method of avoiding this enormous search, let's first understand how the decoder should decode.

To do this, let's see how to generalize the insight from the repetition code. We can think of the repetition code as having two codewords $0,0,0,0,\ldots,0$ and $1,1,1,1,\ldots,1$. The majority rule can be interpreted as asking which one of these is closer to the received string $Y_1, Y_2, Y_3, Y_4, \ldots, Y_n$. This makes sense because if we XOR the received string with the true codeword, we just get back the noise string $Z_1, Z_2, Z_3, Z_4, \ldots, Z_n$. What we are doing is counting the hypothetical number of 1s in this conjectured noise string, and then saying that because the noise prefers to be 0 over being 1, we should pick the codeword that implies a smaller count of 1s for the noise.

There is no reason why in principle we cannot apply this exact same idea to the case of more than 2 messages. We simply decode to the codeword that is closest to the received string in terms of the number of positions in which it agrees with it. If there is a tie, we can just guess or give up.

This isn't a very practical approach since it requires $O(mn)$ operations to do and m might be quite large. But it is certainly doable in principle.

The random coding construction

Shannon's brilliant move was to hope that instead of searching for a needle in a haystack, we were searching for hay in a haystack.

Instead of trying to calculate the probability of error for each message in each codebook one by one, he suggested that we calculate the average probability of error across all messages in all codebooks! At first glance, this seems crazy. If it is hard to calculate it for one, why would it be easier to calculate the average?

The key insight is that by putting a uniform distribution over all messages and all codebooks, we have added a lot of symmetry to the problem. Furthermore, a uniform distribution on all codebooks is the same as putting a uniform distribution on all 2^{mn} bitstrings of length mn which is the same as having all those codebits in the codebook be drawn i.i.d. Bernoulli-$\frac{1}{2}$. This means that each codeword is independent of all other codewords.

Bounding the probability of error

We want to compute the $\mathrm{E}(P(\hat{S} \neq S))$ where the expectation is taken over all the uniform choices for true messages S and codebooks, and the probability inside just has the randomness of the noise in it. It is easy to see[6] that this is the same as $P(\hat{S} \neq S)$ where we allow all the randomness together.

The challenge is that it is hard to exactly analyze even this probability of error. However, the spirit of the laws of large numbers tells us that most of the time, the true codeword will be about np bitflips away from the received sequence. So we can bound the probability of error by thinking about what could go wrong. We could have an error because the noise itself was particularly strange and flipped way more than np bits. Or we could have an error because we got unlucky and happened to get pushed by the noise to be too close to one of the other codewords. Let T be the threshold we used to judge the gap between close and far in terms of the number of bit flips.

$$
\begin{aligned}
P(\hat{S} \neq S) &= P(\exists \text{false } \tilde{s} \text{ so that } \sum_{i=1}^{n}(Y_i + X_i(\tilde{s}) \bmod 2) \leq \sum_{i=1}^{n}(Y_i + X_i(S) \bmod 2)) && (6)\\
&= P(\exists \text{false } \tilde{s} \text{ so that } \sum_{i=1}^{n}(Y_i + X_i(\tilde{s}) \bmod 2) \leq \sum_{i=1}^{n} Z_i) && (7)\\
&\leq P((\sum_{i=1}^{n} Z_i \geq T) \vee (\exists \text{false } \tilde{s} \text{ so that } \sum_{i=1}^{n}(Y_i + X_i(\tilde{s}) \bmod 2) \leq T)) && (8)\\
&\leq P(\sum_{i=1}^{n} Z_i \geq T) + P(\exists \text{false } \tilde{s} \text{ so that } \sum_{i=1}^{n}(Y_i + X_i(\tilde{s}) \bmod 2) \leq T). && (9)
\end{aligned}
$$

[6] Because the inside probability can itself be turned into the expectation of an indicator random variable W that indicates an error, and $\mathrm{E}(\mathrm{E}(W)) = \mathrm{E}(W)$.

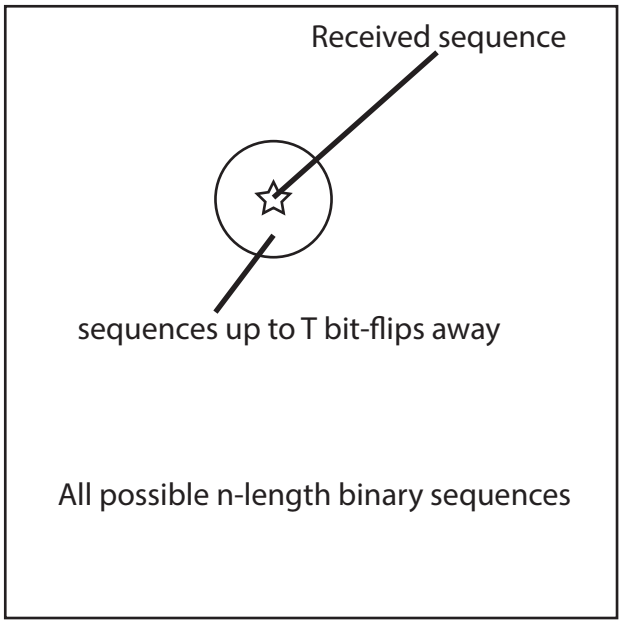

Figure 2: A caricature of the decoder's perspective. It sees the received sequence and basically looks for a codeword nearby (less than T flips away). If the true codeword is not there, then we can declare error. If any false codeword randomly wanders into the circle, then we can declare error. False codewords are uniformly distributed across all sequences and so the ratio of the area of the circle to the area of the square represents the probability of any given false codeword causing trouble by bad luck.

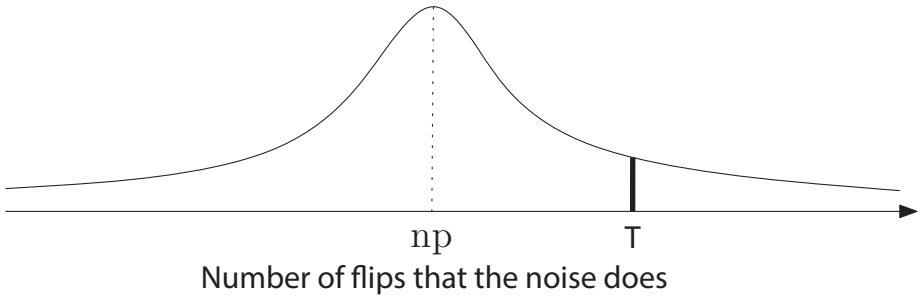

Figure 3: A caricature of how the number of bitflips that the noise does are distributed and where the threshold T is chosen to be.

Here we use capital letters $X_i(\tilde{s})$ to remind ourselves that these codewords are randomly generated. However, notice that the received Y_i depend only on the noise and the true codeword. They are hence independent of all of the false codewords.

Consequently, for purposes of analysis, we can change the order of how randomness is realized in time. Rather than thinking of the false codewords as being generated first, we can think of them as being randomly drawn after the Y_i have been realized. Since they are independent, before and after doesn't matter. But in this changed perspective, we think about the probability that a false codeword wanders too close to the received string. See Figure 2.

Since each false codeword is identically distributed and there are $(m-1)$ of them, we can easily apply the union bound:

$$P(\exists\text{false } \tilde{s} \text{ so that } \sum_{i=1}^{n}(Y_i + X_i(\tilde{s}) \bmod 2) \leq T) \leq (m-1)P(\sum_{i=1}^{n}(Y_i + X_i(f) \bmod 2) \leq T) \tag{10}$$

where $X_i(f)$ are simply Bernoulli-$\frac{1}{2}$ random variables. Let $\|\text{“close”}\|$ be the number of strings $\vec{x}(f)$ that are no more than T bit-flips away from $\vec{Y} = (Y_1, Y_2, \ldots, Y_n)$. It is obvious from the nature of binary strings that $\|\text{“close”}\|$ does not depend on $\vec{Y}$ since no binary string is special when it comes to flipping bits.

Since the $\vec{X}(f)$ are uniformly distributed over all 2^n possible strings, we know

$$P(\sum_{i=1}^{n}(Y_i + X_i(f) \bmod 2) \leq T) = \frac{\|\text{“close”}\|}{2^n}. \tag{11}$$

All that remains is to upper bound $\|\text{“close”}\|$. But this is something that we have seen before:

$$\|\text{“close”}\| = \sum_{k=0}^{T}\binom{n}{k}.$$

To finish, let us set $T = n(p+\varepsilon)$ for some $\varepsilon > 0$ so that $0 \leq p < p+\varepsilon < \frac{1}{2}$. See Figure 3. Then we know by the law of large numbers[7] that the first term in (9) will go to zero as $n \to \infty$.

[7] In fact by using the Chernoff bound:

$$P(\sum_{i=1}^{n} Z_i \geq T) \leq e^{-nD(p+\varepsilon||p)}$$

where $D(p+\varepsilon||p) > 0$. A second-order Taylor expansion of D about p will reveal the basic speed. It will show that if we want to keep the probability of this kind of error small, we just need to choose $\varepsilon = \frac{K}{\sqrt{n}}$ for some K that depends on both p and the desired probability of error. See the note on the central limit theorem to understand this better.

The exact speed doesn't matter. What matters is

$$\begin{aligned}
\|\text{“close”}\| &= \sum_{k=0}^{T} \binom{n}{k} \\
&\leq (T+1) \max_{0\leq k\leq T} \binom{n}{k} \\
&= (T+1)\binom{n}{T} \\
&= (n(p+\varepsilon)+1)\binom{n}{n(p+\varepsilon)} \\
&\approx (n(p+\varepsilon)+1)2^{n(H(p+\varepsilon))} \\
&\approx 2^{n(H(p+\varepsilon))}
\end{aligned}$$

where the first inequality is clear since replacing each term in a sum by the biggest term always gives something bigger. The fact is that $\binom{n}{k}$ gets bigger with k as long as $k < \frac{n}{2}$. The first approximation comes from notes on fair coin tosses where we calculated using Stirling's approximation that $\binom{n}{np} \approx 2^{nH(p)}$ where $H(p) = p\log_2 \frac{1}{p} + (1-p)\log_2 \frac{1}{1-p}$ is the binary entropy function. The second approximation merely recognizes that an exponential is much bigger than a polynomial.

Combining the above with (11) and (10) gives us:

$$\begin{aligned}
P(\exists\text{false } \tilde{s} \text{ so that } \sum_{i=1}^{n}(Y_i + X_i(\tilde{s}) \bmod 2) \leq T) &\leq (m-1)\frac{\|\text{“close”}\|}{2^n} \\
&\approx m2^{nH(p+\varepsilon)}2^{-n} \\
&= m2^{n(H(p+\varepsilon)-1)}
\end{aligned}$$

This will go to zero as $n \to \infty$ as long as m grows more slowly than $2^{n(1-H(p+\varepsilon))}$. Consequently by continuity of $H(p)$ in p, for any rate $R < 1 - H(p)$, there exists an $\varepsilon > 0$ so that we can show that both of the error terms in (9) go to zero as $n \to \infty$.

So the average probability of error goes to zero as we let n get large as long as the rate of the codebook $R < 1 - H(p)$. For $p = 0.05$, this corresponds to rates less than 0.71 or so. (Far higher than than the 0.14 and the like that we were getting before.)

From averages to existence

So we know that on average over codes and messages, we do fine. But does that mean that any actual code exists that protects all possible messages that we might want to send?

At this point, we simply apply Markov's inequality twice. Once over codes and second over messages:

1. If codes on average have good performance, that must mean that there exist many codes (i.e. half of all possible codes) whose average performance has a probability of error at most twice the average for all possible codes. (Markov's inequality). But this average is tending to zero as n gets large and so then is twice the average.

2. For any of these particular codes, at least half the codewords must have an average probability of error that is at most twice the average for this code. (Markov's inequality again). But again, since the average is tending to zero as n gets large, so is twice the average.

3. Now we know that there are lots of codes for which the best half of the codewords have a probability of error that is tending to zero. So, just expunge the worst half of the codewords in any of these codes! This gives a new codebook with half the number of messages. The elimination of possibly confusing codewords can only reduce the probability of error for any surviving codeword. But now every codeword in the new codebook has an average probability of error at most 4 times the average over all possible codebooks and messages computed in the random coding argument above.

4. This "expurgation" costs at most 1 bit worth of messages since a factor of 2 in m is only 1 bit in terms of the binary representation of messages. So plenty of good codes must exist at rates strictly less than $1 - H(p)$ bits/raw-bit, as long as n is made large enough.

Notice the amazing thing here. We have shown that good codes exist — dramatically better than repetition codes. In fact, we have shown that digital communication is possible and that reliability comes from "convoying" — having lots of data bits share space in a long codeword where they share redundancy across each of them. The probability of error can be made to go to zero simply by increasing the size of the convoy and making n large. It doesn't need the rate R to go to zero. Instead, the rate R is simply limited by the average number of flips. If there are lots of flips on average, p is closer to $\frac{1}{2}$ and $1 - H(p)$ is closer to zero.

This idea that the only fundamental cost of reliability was solidarity among individual bits prompted the search for practical codes to see if this was actually possible in practice and not just in theory. This laid the foundations for the modern digital age. But without knowing that good codes existed in principle, nobody would even have begun searching for them. And proving their existence required probability.

If you want to learn more about such arguments, take EECS 121 and then EECS 229A. The ideas here turn out to generalize and also talk about data compression. For more on the practical codes that result, they are touched upon in EECS 121 and in much greater depth in EECS 229B. This general style of using probability to prove existence is used in EECS 174 in graph contexts, and then in advanced graduate courses on algorithms.

The kinds of random structures found for codes are also used a lot in AI algorithms, and increasingly for signal processing applications as well.