Error correction

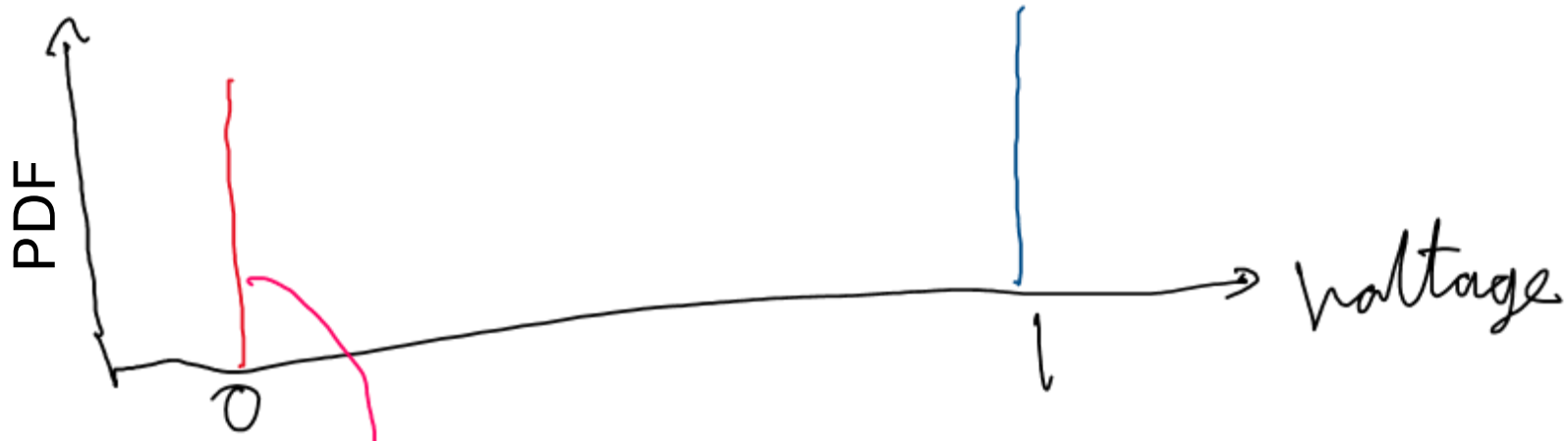
Lecturer: Venkat Arun

Logistics

- Quiz will be in the next class. It will include everything up to and including this class.
  - You are allowed 3 pages of notes
  - You can ask me and Jeongyoon any *specific* and *factual* questions you like *during* the quiz, and we will answer them! You are responsible only for reasoning, not knowing facts. Examples of factual questions include details of how an algorithm works, any formulas, and definitions of terms
- Lectures are recorded and available on canvas
- Slides are available on the course page
- From yesterday:
  - Hamming codes from yesterday are not a part of the syllabus
  - We'll skip reliable transmission beyond what we discussed in class yesterday, and continue it when we discuss the transport layer

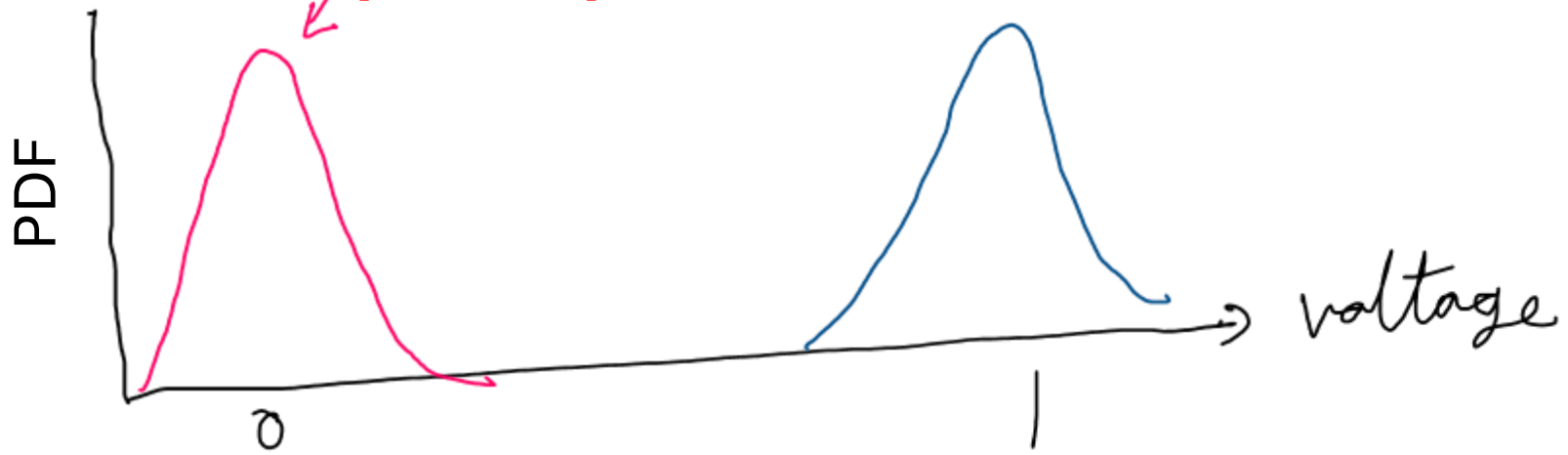
Simple model for why errors happen

What sender sent



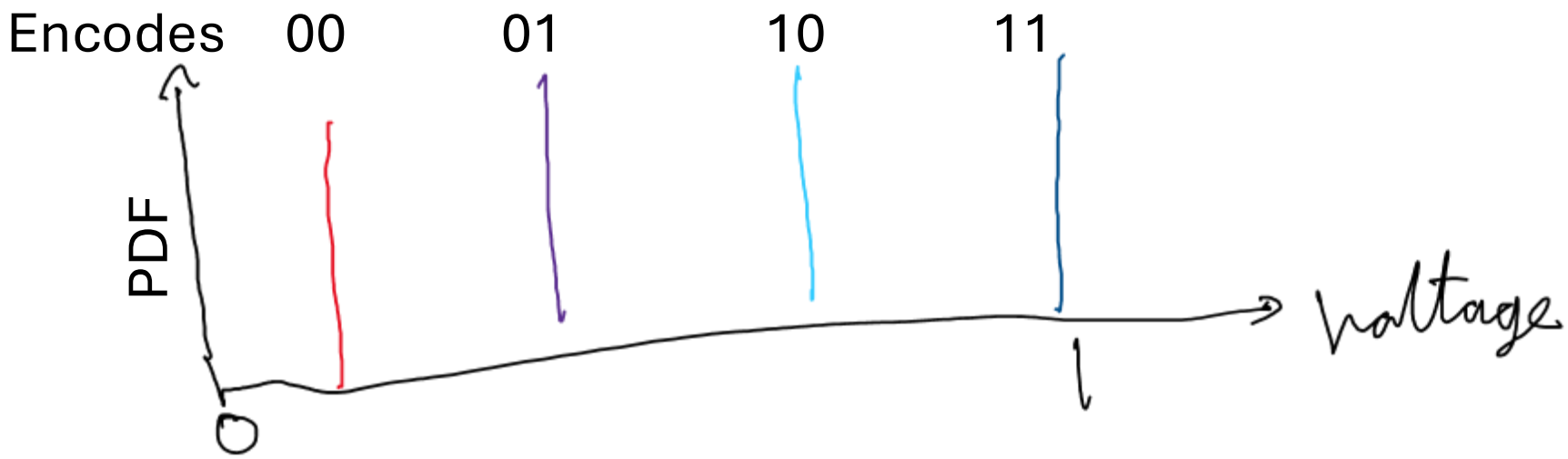
0V becomes approx.
[-0.25, 0.25] V

What receiver got (probability density function)

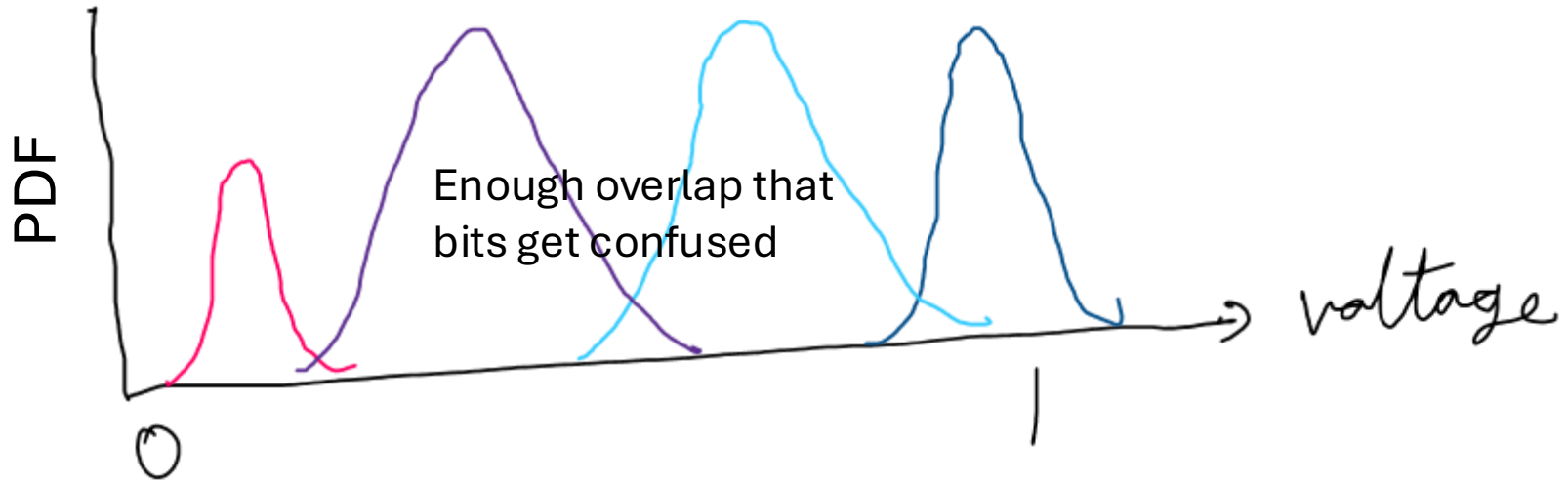


Simple model for why errors happen

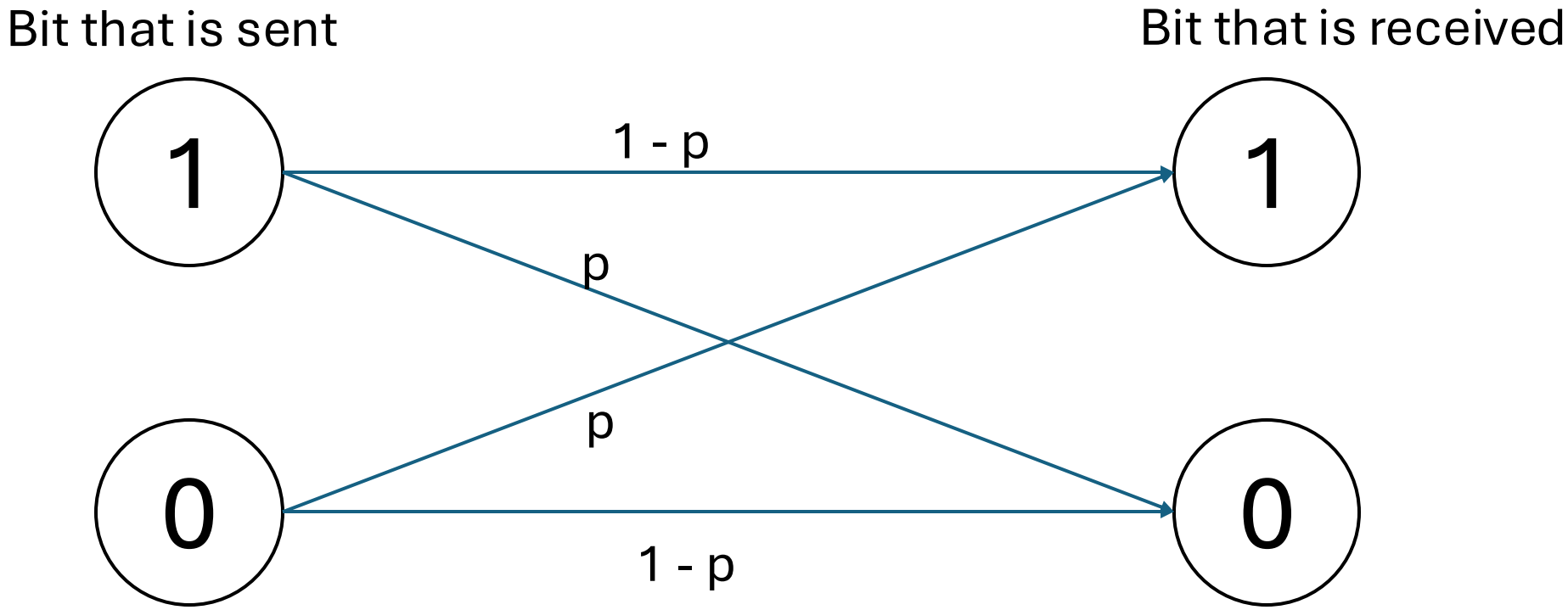
What sender sent



What receiver got (probability density function)

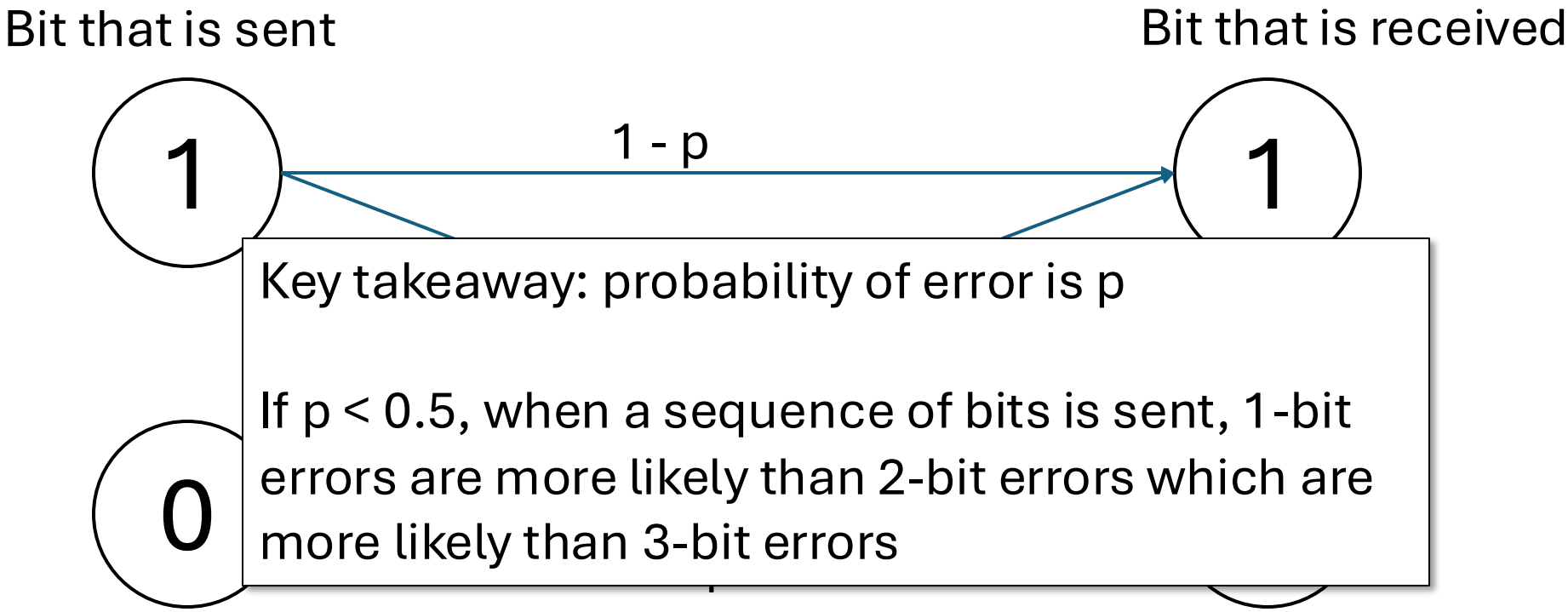


What errors look like in the digital world



The values on the edges denote the probability that the transmitted bit is received as a 0 or a 1

What errors look like in the digital world



The values on the edges denote the probability that the transmitted bit is received as a 0 or a 1

Example of an error correcting code: triple repetition

Data I want to transmit	How I encode the data
0	000
1	111

What I receive	What I interpret it as
000	0
111	1

Example of an error correcting code: triple repetition

Data I want to transmit	How I encode the data
0	000
1	111

What I receive	What I interpret it as
000	0
111	1
001	??

Example of an error correcting code: triple repetition

Data I want to transmit	How I encode the data
0	000
1	111

What I receive	What I interpret it as
000	0
111	1
001	0
011	??

Example of an error correcting code: triple repetition

Data I want to transmit	How I encode the data
0	000
1	111

What I receive	What I interpret it as
000	0
111	1
001	0
011	1
101	1
100	0

A longer error correcting code

Data I want to transmit	How I encode the data (codeword)
0 0 0 0	0 0 0 0 0 0 0
0 0 0 1	1 1 0 1 0 0 1
0 0 1 0	0 1 0 1 0 1 0
0 0 1 1	1 0 0 0 0 1 1
0 1 0 0	1 0 0 1 1 0 0
0 1 0 1	0 0 1 1 0 0 1
0 1 1 0	1 1 0 0 1 1 0
0 1 1 1	0 0 0 1 1 1 1
1 0 0 0	1 1 1 0 0 0 0
1 0 0 1	0 1 0 0 1 0 1
1 0 1 0	1 0 1 1 0 1 0
1 0 1 1	0 1 1 0 0 1 1
1 1 0 0	0 1 1 1 1 0 0
1 1 0 1	1 0 1 0 1 0 1
1 1 1 0	0 0 1 0 1 1 0
1 1 1 1	1 1 1 1 1 1 1

Points to note

- Every bit-string on the right differs from every other bit-string by at least 3 bits. This is called as having a “hamming distance” of at least 3
- This means that if one bit of any codeword is flipped, its distance to every other codeword is >= 2. Its distance to the original codeword is of course 1. Thus, it can be decoded
- If two bits are flipped, it cannot be **corrected**, but the error can be **detected** since no two codewords differ by 2 bits

- How do we come up with this table? This is called a hamming code and is out of scope of this course. It is simple enough to understand though

A longer error correcting code

Data I want to transmit	How I encode the data (codeword)
0 0 0 0	0 0 0 0 0 0 0
0 0 0 1	1 1 0 1 0 0 1
0 0 1 0	0 1 0 1 0 1 0
0 0 1 1	1 0 0 0 0 1 1
0 1 0 0	1 0 0 1 1 0 0
0 1 0 1	0 0 1 1 0 0 1
0 1 1 0	1 1 0 0 1 1 0
0 1 1 1	0 0 0 1 1 1 1
1 0 0 0	1 1 1 0 0 0 0
1 0 0 1	0 1 0 0 1 0 1
1 0 1 0	1 0 1 1 0 1 0
1 0 1 1	0 1 1 0 0 1 1
1 1 0 0	0 1 1 1 1 0 0
1 1 0 1	1 0 1 0 1 0 1
1 1 1 0	0 0 1 0 1 1 0
1 1 1 1	1 1 1 1 1 1 1

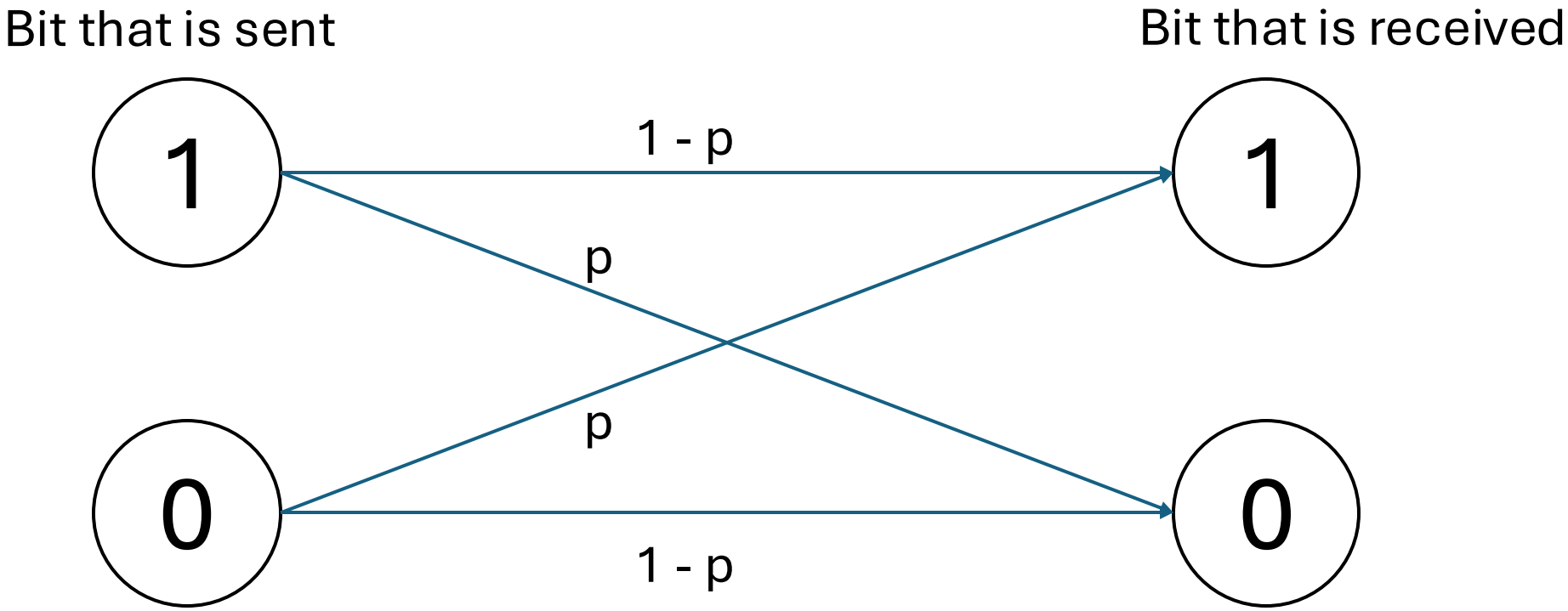
Here, we are encoding 4 bits as 7 bits

In general, for all m, you can encode $2^m - m - 1$ bits using a codeword of $2^m - 1$ bits. In each case, you can correct 1 bit errors and detect 2 bit errors

For example:
1 bit → 3 bits (efficiency: 0.333, m = 2)
4 bits → 7 bits (efficiency: 0.571, m = 3)
11 bits → 15 bits (efficiency: 0.733, m = 4)
16 bits → 31 bits (efficiency: 0.839, m = 5)

Key takeaway: To correct/detect a given number of bit errors, the longer codewords you use, the greater the efficiency.
Efficiency approaches 1 as codeword length goes to infinity!

Multi-bit correction: How much error correction do we need?



If probability of error is p, then we will have $\approx Kp$ errors in every K bits

Magic of statistics: as $\mathrm{K} \to \infty$, the number of errors will be almost *exactly* Kp

In case you are interested, this is called the law of large numbers

Multi-bit correction: How much error correction do we need?

If probability of error is p, then we will have $\approx Kp$ errors in every K bits

Magic of statistics: as $\mathrm{K} \to \infty$, the number of errors will be almost *exactly* Kp

In case you are interested, this is called the law of large numbers

Task: To transmit N bits, we need to find a $\mathrm{K} > N$ bit code book (set of codewords) that encodes K bits such that we can correct Kp. Efficiency is N / K

Shannon's theorem (informally stated): For every channel with probability of error p, we can construct bigger and bigger code books (i.e. larger and larger N and K) such that the efficiency approaches 1 – H(K). That is, $\frac{N}{K} \to 1 - H(p)$ as $N \to \infty$

Here, $H(p) = -p \log p - (1-p) \log(1-p)$ is called "information entropy"

Multi-bit correction: How much error correction do we need?

Task: To transmit N bits, we need to find a K $> N$ bit code book (set of codewords) that encodes K bits such that we can correct Kp. Efficiency is N / K

Shannon's theorem (informally stated): For every channel with probability of error p, we can construct bigger and bigger code books (i.e. larger and larger N and K) such that the efficiency approaches 1 – H(K). That is, $\frac{N}{K} \to 1 - H(p)$ as $N \to \infty$

Here, $H(p) = -p \log p - (1-p) \log(1-p)$ is called "information entropy"

Strategy: Pick a very large K and pick $N = \big(1 - H(p)\big)K$

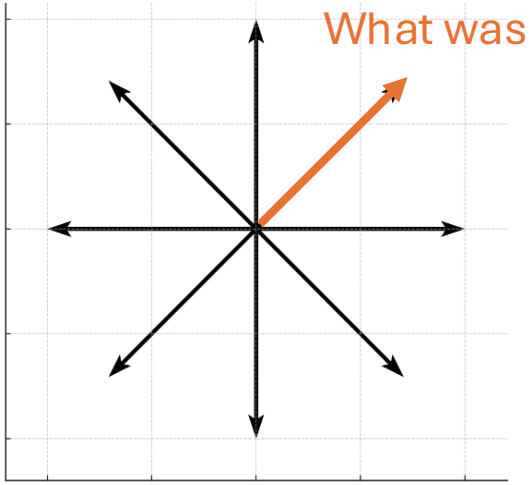
Key takeaways

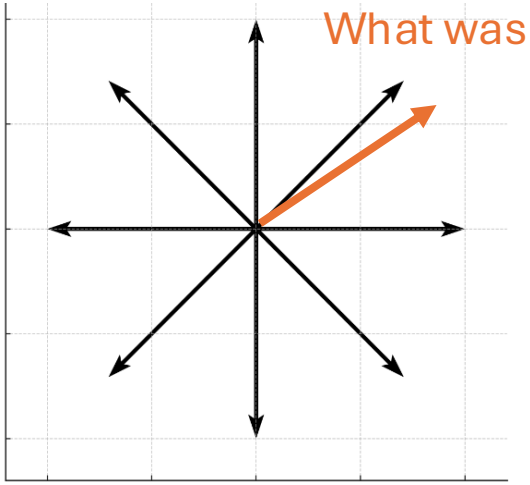
The following is all I expect you to understand about multi-bit error coding:

- We can encode messages by their corresponding codewords
- A bit-flip channel flips the bit with probability p
- For p, there is a maximum achievable efficiency for messages sent through that channel.
- That is, we can encode N-bit messages using K-bit codewords such that N / K approaches this maximum as $N \to \infty$

Fun aside: Analogy between neural networks and Shannon's code

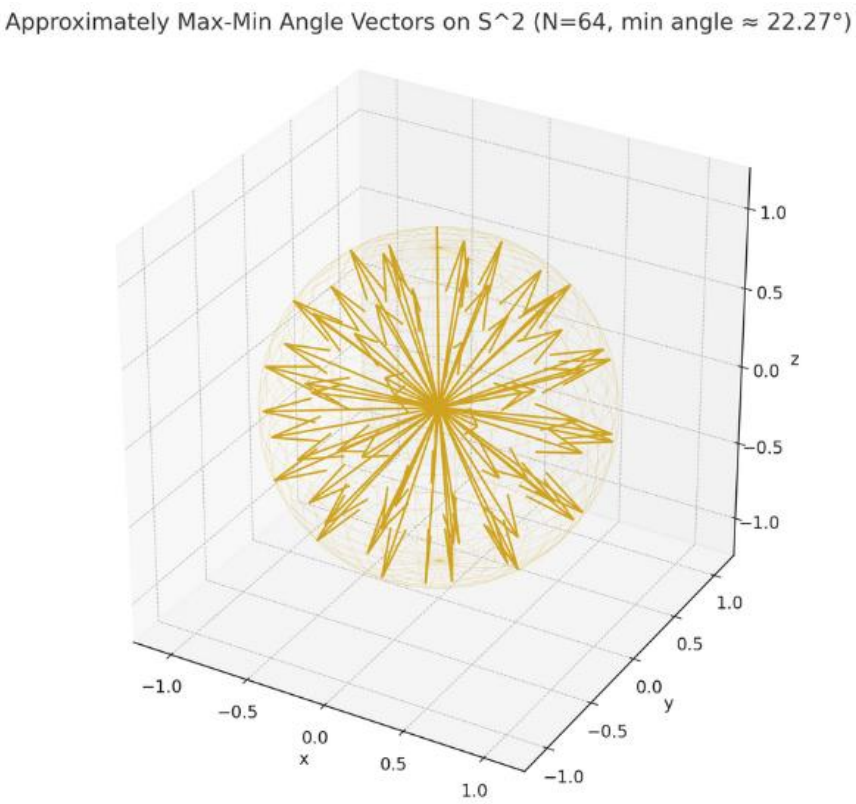
- Say we transmit vectors and what is received is the transmitted vector rotated by a little bit (this is not how physical channels work, but in high dimensions, it is close enough)
- The transmitter encodes information in one of the 8 available vectors ($\log_2 8 = 3$ bits)
- The receiver corrects to the nearest vector. If the noise is less than $22.5°$, the result will be correct





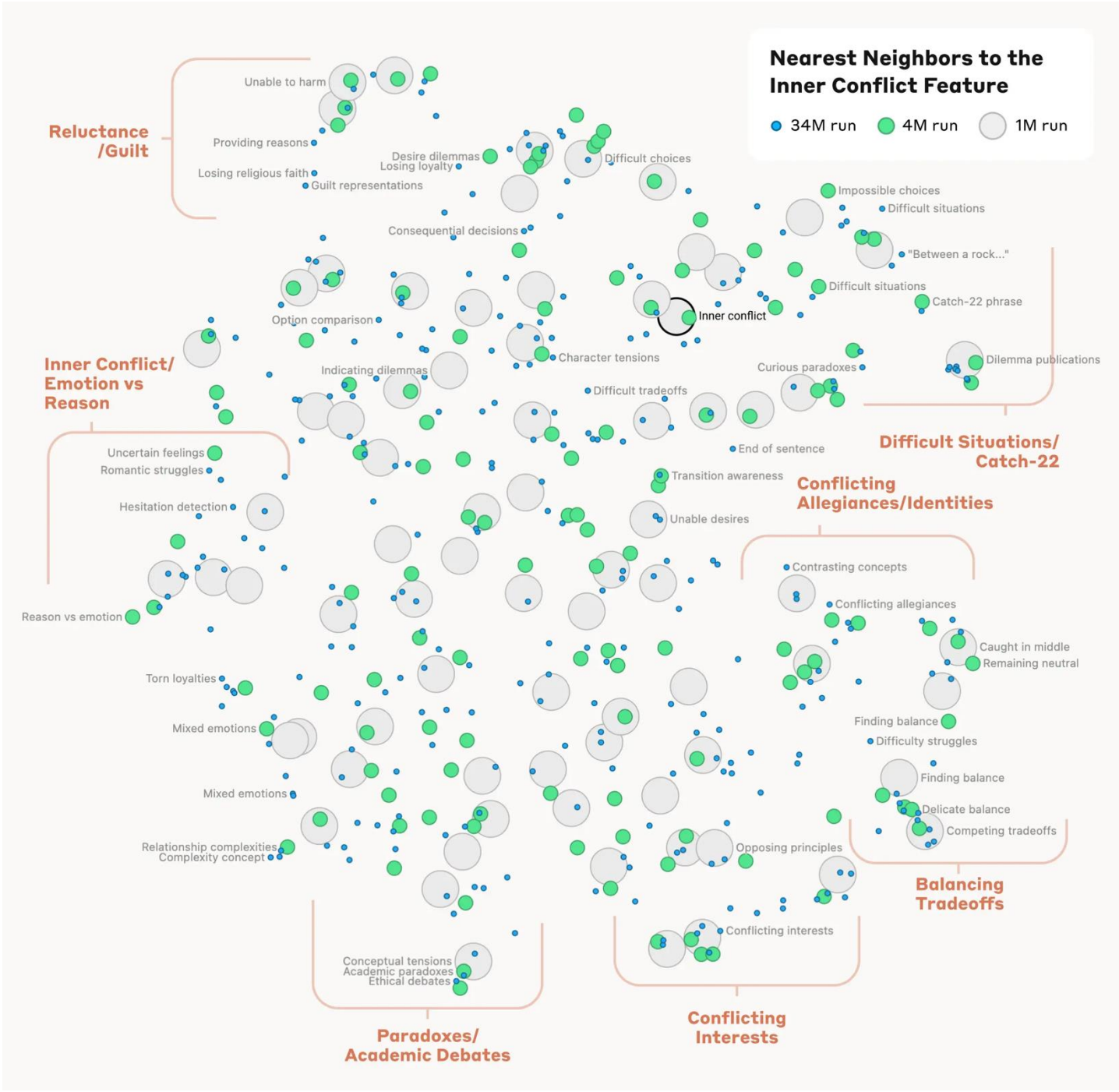
Fun aside: Analogy between neural networks and Shannon's code

- In higher dimensions, you can fit more vectors while still making sure the angle between any two vectors is larger than the noise.
- In general, in a d-dimensional space, you can pick $O(2^d)$ unit vectors (and hence encode $O(d)$ bits) such that the distance between any two of them is $O\left(\frac{1}{\sqrt{d}}\right)$
- How do we pick these vectors? Easy, just pick randomly!



Fun aside: neural networks

- Many neural networks have been shown to represent concepts as directions in a high dimensional vector space.
- Here is an excellent study by Anthropic that interprets LLMs' internal representation using this idea: https://www.anthropic.com/research/mapping-mind-language-model
- If the vector has $O(d)$ dimensions, the network can represent $O(2^d)$ concepts!
- Anthropic found directions corresponding to many high-level concepts (see right)



Fun aside: Shannon's code

- A similar result is true if, instead of using real numbers, the vectors are in the field Z_2 i.e., all elements are binary.
  - A vector is a bit-string. The sum of two vectors is a bit-wise XOR of the two strings.
  - The distance (dot product in the previous slide) between two vectors is the Hamming distance between them (i.e. the number of positions where the bit strings are different)
- Thus, you can have a large number of vectors (bit-strings) that have a large hamming distance between them (say X). This is exactly what we needed for error correction. As long as < X/2 bits are flipped, we can correct the error
- The larger K needs to be, the fewer vectors (bit-strings) you can pack before you get a pair that differ by < X bits.

Recap: How do we correct more bit errors?

- *Shannon's code:* For every N-bit message, pick a *random* K-bit string as the codeword. Let this mapping from message to codeword be C(m). The sender and receiver need to agree on this function

- Decoding strategy: given a received value of x, find a message m such that the hamming distance between x and C(m) is minimized. m will be the most likely transmitted message

- What is the computational complexity of this decoding process?

Fun aside: How do we correct more bit errors?

- What is the computational complexity of this decoding process?

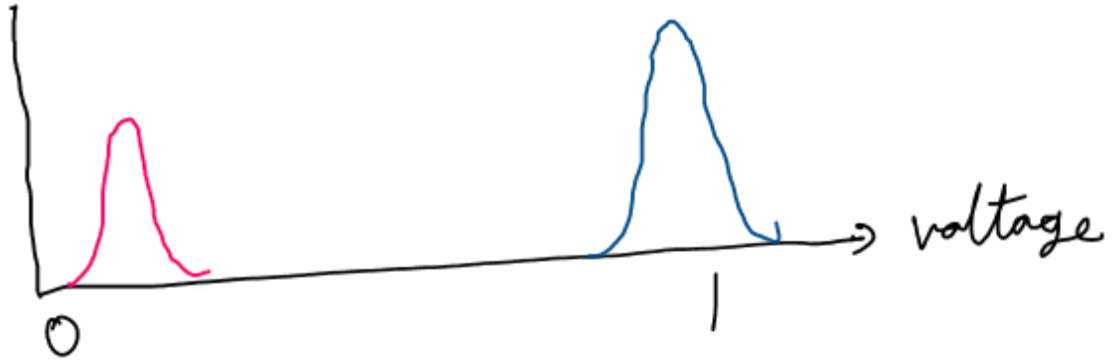
- Possible strategy to decode received message x:
  - I received x
  - Is$C^{-1}(x)$a message? If so, return $C^{-1}(x)$
  - For all integers i
    - For all x' such that hamming distance between x and x' is i Complexity: there are 2^i such x'!
      - Is $C^{-1}(x)$ a message? If so, return $C^{-1}(x)$

Key takeaways and some history

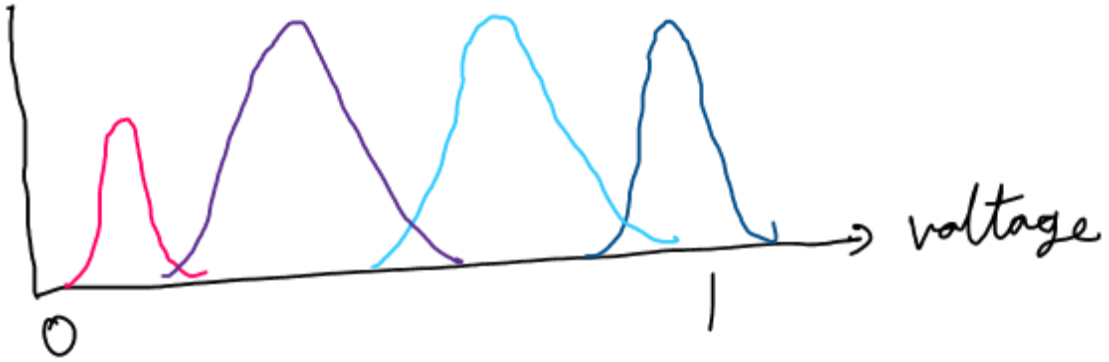
- Decoding Shannon codes has exponential complexity
- Shannon published his ground-breaking work, including the Shannon code, in 1948
- However, for a long time we did not have a coding scheme that approached the theoretical limit *and* could be decoded in efficiently (note the two different types of efficiency here).
- In the 1990s, we started studying the first solutions to this problem (called Turbo codes and LDPC codes)
- This revolutionized cellular and WiFi networks because, for the first time, they were able to approach the theoretical optimal message transmission rates! This was quickly adopted everywhere
- Side note: Turbo codes were patent protected until a few years ago. This made LDPC codes more popular in some places

Key takeaways and some history

Thanks to these new coding schemes, we could start packing bits more tightly. This way, errors are more likely to happen, but we now have efficient methods to correct them. This is what allowed wireless networks to achieve greater efficiency



More conservative encoding that prevented errors



Tighter, more aggressive encoding