

Lecture 15: Convolutional codes

ENGR 76 lecture notes — May 19, 2026

David Tse¹, Stanford University

In this lecture, we introduce a new family of codes called convolutional codes that are widely used in deep space communication and Wi-Fi. Convolutional codes are also the building blocks of more advanced codes like turbo codes, which combines two convolutional codes, and are used in space communication, 3G and 4G. We will also use convolutional codes in the audio communication project.

1 Short Versus Long Block Lengths

All the codes we have seen till now have short block lengths. For example, the Hamming code can be thought of as a mapping from 4 information bits to 7 coded bits, defined using the parity equations or through a lookup table. It has a block length of 7. To encode a long sequence of information bits with the Hamming code, we first need to chop the sequence into blocks of 4 bits and then apply the encoding to each block separately. For example, if the sequence of information bits was 101011100011, then we would first split it into 3 blocks 1010, 1110, 0011. Then we would apply the encoding to each block to get 1010010, 1110100, 0011110. Finally we would concatenate the encoded blocks to get the final encoding 101001011101000011110. Since the Hamming code is capable of correcting one bit flip, block coding in this manner can tolerate up to one bit flip per block.

Next, we look into convolutional codes, which can have arbitrarily long block lengths without chopping the input sequence into smaller blocks.

2 Convolutional codes

We'll focus on a specific convolutional code in this lecture, study its encoding and view the code through a few different perspectives.

2.1 Encoding

The encoder takes a binary information sequence $b_1, b_2, \dots, b_k \in \{0, 1\}$ and generates the encoded sequence $x_1, x_2, \dots, x_n \in \{0, 1\}$. In general, we'll have $n > k$, for this specific code $n = 2k$ and the encoded sequence is obtained as follows:

For $j = 1, \dots, k$:

$$p_j^{(1)} = b_j \oplus b_{j-1} \oplus b_{j-2}$$

$$p_j^{(2)} = b_j \oplus b_{j-2},$$

so that the encoded sequence satisfies $x_{2j-1} = p_j^{(1)}$ and $x_{2j} = p_j^{(2)}$ for $j = 1, \dots, k$.

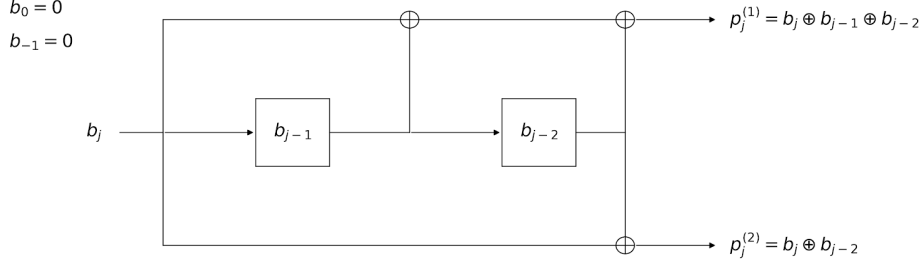
Recall that the XOR or modulo 2 sum operation (denoted by $\oplus$) simply counts the number of 1s and results in 0 if the number is even, and 1 otherwise. For example, $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$. The operation is commutative and associative, and hence can be computed in any order.

In the expression above, we assume that $b_0 = b_{-1} = 0$, which can be thought of as the initial condition of the system. Note how the encoding happens in a streaming fashion - at each step j , the encoder takes b_j and produces $p_j^{(1)}$ and $p_j^{(2)}$ based on b_j , b_{j-1} and b_{j-2} .

¹These notes are adapted from the 2025 Spring offering by Ayfer Özgür

Note that each of the information bits $b_1, b_2, \dots, b_{k-2}$ influences 5 coded bits, which provide its error protection. However, the last two bits b_{k-1} and b_k influence less coded bits, so they receive less protection. To restore the symmetry between the information bits, we will pad the information bits with two zeros, so that the input to the convolution encoder is $b_1 b_2, \dots, b_k 00$, resulting in the output $p_1^{(1)} p_1^{(2)} \dots p_{k+2}^{(1)} p_{k+2}^{(2)}$. So the rate of this code is not exactly $1/2$ but $k/(2k+4)$. For k large, which is typically the case in practice, the rate is very close to $1/2$.

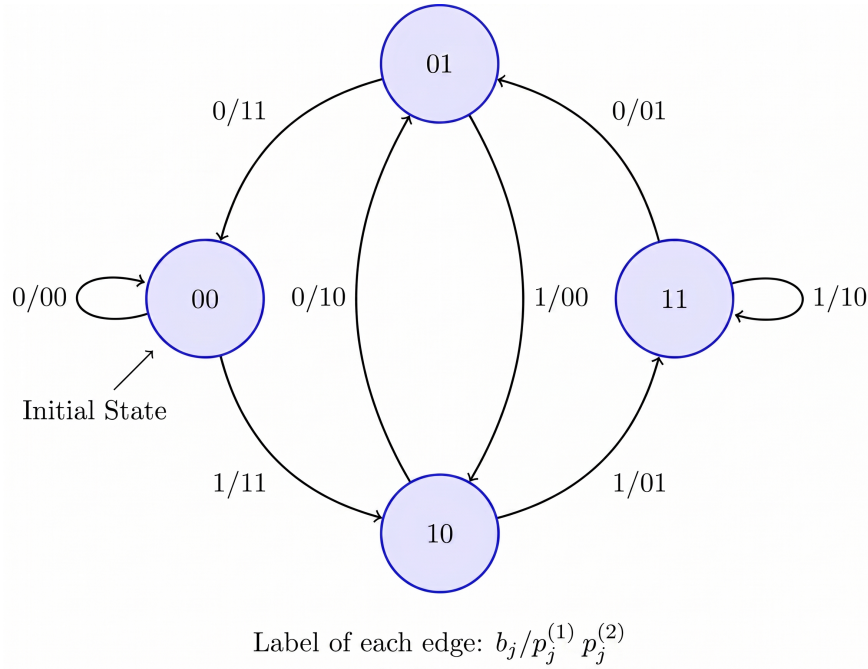
2.2 Shift register view



Next we see the encoding in a different way that is closely related to how the encoding works in hardware. As seen in the figure above, the shift register stores the previous two bits (b_{j-1}, b_{j-2}) in memory, and after each step the new symbol b_j pushes b_{j-1} to the right, and b_{j-2} is removed from the memory (the *shift* operation). At step j , the outputs are computed with the $\oplus$ gate as shown in the figure. Initially, both memory elements store 0.

We can consider an example for the encoding. Say the information bit sequence is 01101. We pad it with two extra zeros as above: 0110100. Then the encoding proceeds as follows. At step 1, we have $b_{j-1} = b_{j-2} = 0$ and $b_j = 0$. Thus, both output bits $p_1^{(1)} = b_j \oplus b_{j-1} \oplus b_{j-2}$ and $p_1^{(2)} = b_j \oplus b_{j-2}$ are 0. After the shift operation, we have $b_{j-1} = b_{j-2} = 0$ and the new bit $b_j = 1$ (where $j = 2$ now). Now we get $p^{(1)}_2 = p^{(2)}_2 = 1$. After the shift operation, we have $b_{j-1} = 1, b_{j-2} = 0$ and the new bit $b_j = 1$ (where $j = 3$ now). Then we get $p^{(1)}_3 = 1 \oplus 1 \oplus 0 = 0$ and $p^{(2)}_3 = 1 \oplus 0 = 1$. The process continues until we are done with the input bits. Note that because of the padding, at the end of the process, the register contains two zeros.

2.3 State machine view



The encoding process can be considered as a finite state machine, where we have multiple states, with each input bit leading to a transition from one state to another along with producing some output bits. The *state* of the convolutional code is simply the memory in the shift register, i.e., state is $b_{j-1}b_{j-2}$ (note that the order of bits in the state is opposite the order of bits in the input, we use this convention to match the shift register view). The initial state is 00. The total number of states is just the number of values $b_{j-1}b_{j-2}$ can take, which is 4. The transitions are labeled with the input bit causing the transition and the output bits produced during the transition, denoted as $b_j/p_j^{(1)} p_j^{(2)}$.

Let's try to understand some of the transitions. If the current state is $b_{j-1}b_{j-2} = 00$, and the input bit is $b_j = 0$, then we can verify that the output bits are 00 and the next state is 00 (this is simply $b_j b_{j-1}$). This is shown as the self loop at the 00 state on the left. If the current state is $b_{j-1}b_{j-2} = 00$, and the input bit is $b_j = 1$, then we can verify that the output bits are 11 and the next state is 10 (this is simply $b_j b_{j-1}$). This is the arrow going from 00 to 10. Similarly the rest of the state diagram can be obtained. As we get input bits, we move from one state to another, and keep on producing output bits as shown in the diagram. Note that because of the padding at the end, we always return to the 00 state,

2.4 Minimum distance

We learnt in the last lecture that the minimum distance $d_{\min}$ of a code determines the maximum number of errors the code can correct. So what is the minimum distance of the convolutional code considered in this lecture? Although the minimum distance is a minimization over all pairs of codewords, it can be shown that for convolutional codes, due to its linearity, it suffices to find the minimum distance between a codeword and the all-zeros codeword, i.e. find the minimum weight among all non-zero codewords, i.e. the non-zero codeword with the minimum number of 1's. We can find this using the finite state machine representation of the code. The all zero codeword is one in which the finite state machine remains in the 00 state forever. A nonzero codeword starts from the 00 state, departs from it at some point and then comes back to it. Note that to depart from 00, we must transition to 10. This will output 11, incurring a weight of 2. To go back to the 00 state as quickly as possible, we should go to 10 and then return to 00. This incurs any extra weight of $1 + 2 = 3$, resulting in a total weight of 5. Hence, the minimum weight among all non-zero codewords is 5, thus this code can correct 2 errors. (Please convince yourself this is the path of least weight!)

2.5 Constraint Lengths and Code Rates

There are many different convolutional codes, of which we just gave one example above. The two key parameters of a convolutional code are : 1) the constraint length K : how big is the window of information bits a coded bit depends on ; 2) code rate: the number of information bits sent per coded bits. The example convolutional code has constraint length $K = 3$ and code rate $1/2$. In general, the number of states is 2^{K-1} , so the larger the constraint length, the larger the minimum distance can be. For example, the convolution code used in the Opportunity Mars Rover has constraint length $K = 15$, code rate $1/6$ and $d_{\min} = 57$, correcting 28 errors.