Implementation of Polar Codes Using Arıkan's Method

Isaiah Gocool
Graduate Student, Department of Electrical and Computer Engineering
York University
Toronto, Canada
goisaiah@my.yorku.ca

***Abstract*—Polar codes, introduced by Erdal Arıkan in 2009, were the first codes to provably achieve the channel and Shannon capacity for a binary-input discrete memoryless channels under low-complexity successive cancellation (SC) decoding [1]. In this paper, we have implemented a polar coding system using a Jupyter notebook written in Python that is based on the work conducted by Arıkan. It includes frozen bit selection based on Bhattacharyya parameters, polar encoding, and SC decoding with the option for exact or min-sum log-likelihood update rules. The notebook also contains Monte Carlo simulations that evaluate the correctness of the code without noise, and tests with noise that demonstrate how the bit error rate (BER) decreases as the block length increases.**

***Index Terms*—Information theory, polar codes, channel polarization, channel capacity, successive cancellation decoding**

I. Introduction

One of the major problems in the field of information theory is the reliable transmission of data over noisy channels. Claude Shannon theorized in 1948 that for a discrete channel with noise, there exists a coding system that can transmit information with an arbitrarily low frequency of errors as long as the transmission rate is below channel capacity [2]. The maximum rate of error-free data that can be transmitted is referred to as the Shannon capacity [2]. Codes achieving channel capacity remained an open problem for many decades, with early work in the field of error-correcting codes pioneered by Richard Hamming through the creation of Hamming codes in 1950 [3], and turbo codes in 1993 which demonstrated a significant improvement but still could not achieve Shannon capacity [4].

This problem was definitively solved by Erdal Arıkan in 2009 with his creation of polar codes. Polar codes are the first practical codes to achieve the Shannon capacity of binary-input discrete memoryless channels (B-DMC) through successive cancellation (SC) decoding [1]. The key process in polar codes that allows them to reach Shannon capacity is channel polarization, which converts $N = 2^n$ independent copies of a channel into sub-channels that are considered either reliable or unreliable depending on the amount of noise they contain. Information bits are placed in reliable sub-channels, while frozen bits are assigned to unreliable sub-channels. Frozen bits provide known constraints that guide the successive cancellation (SC) decoder. As N increases, the fraction of reliable channels approaches the Shannon capacity. This process enables the decoder to progressively recover information with decreasing uncertainty as block length grows.

In this paper, we work to implement polar codes in a Jupyter notebook using Python. Deterministic tests are used to evaluate the correctness of the program, and Monte Carlo tests are used to evaluate empirical errors, such as bit error rate (BER) and block error rate (BLER). The results demonstrate how as N increases, BER decreases and the fraction of reliable channels approaches Shannon capacity.

II. Background and Definitions

A. Binary-Input Discrete Memoryless Channel (B-DMC)

A binary-input discrete memoryless channel (B-DMC) W is a channel with finite input and output alphabets, and its current output is independent of any previous input [1]. The binary input alphabet is $\mathcal{X} = \{0, 1\}$ and the output is $\mathcal{Y}$. The set of transition probabilities is $W(y \mid x)$, which represents the probability of receiving output $y \in \mathcal{Y}$ given the transmitted bit $x \in \mathcal{X}$. The channel corresponding to N uses of W is denoted by W^N, which has its transition probabilities defined by:

$$W^N(y_1^N \mid x_1^N) = \prod_{i=1}^{N} W(y_i \mid x_i)$$

In our Python code, we implemented a special case of a B-DMC known as the binary symmetric channel (BSC), which we defined in the function `bsc`.

B. Channel Capacity

The capacity of a B-DMC is the maximum rate at which information can be reliably transmitted over a channel. Channel capacity is typically expressed using the mutual information as $C = \max_{p(x)} I(X;Y)$ [5]. $I(W)$ denotes the symmetric capacity of a B-DMC, which is the highest rate achievable assuming a uniform input distribution. As is the case with symmetric channels, $C = I(W)$ [1]. The full equation defining $I(W)$ is as follows:

$$I(W) \triangleq \sum_{y\in\mathcal{Y}} \sum_{x\in\mathcal{X}} \frac{1}{2} W(y|x) \log \left(\frac{W(y|x)}{\frac{1}{2}W(y|0) + \frac{1}{2}W(y|1)} \right)$$

For a BSC with crossover probability p, the capacity can also be simplified to $C = 1 - H_2(p)$, where $H_2(p)$ is the binary entropy function. Our code used the functions `binary_entropy` and `plot_reliable_fraction` to calculate the channel capacity.

C. Mutual Information

As mentioned earlier, $I(W)$ is the symmetric capacity of a B-DMC, but it is also the mutual information of a B-DMC W [1]. Mutual information $I(X;Y)$ quantifies the amount of information gained about X from observing Y [5]. This is used as a rate measure for polar codes, since it is the highest rate at which reliable transmission is possible [1].

D. Bhattacharyya Parameter

The Bhattacharyya parameter $Z(W)$ is used as a measure of reliability for binary channels [1]. It provides an upper bound on the probability of a maximum-likelihood (ML) decision error occurring when the channel is used to send either a '0' or a '1' once. For a B-DMC W, it is defined as:

$$Z(W) \triangleq \sum_{y\in\mathcal{Y}} \sqrt{W(y \mid 0)\, W(y \mid 1)}$$

This parameter is essential for the channel polarization result, because $Z(W) \approx 0$ corresponds to a reliable noise-free channel, and $Z(W) \approx 1$ indicates a noisy unreliable channel [1]. Therefore, it helps the polar code system to decide which channel they should transmit information through. The Bhattacharyya parameter is computed in our code through the function `bhattacharyya_parameter`.

E. Information and Frozen Bits

When polar codes are constructed, the original message is represented by the vector $\mathbf{u}$, its block length is represented by $N = 2^n$, and it is divided into two parts based on an information set $\mathcal{A}$. $\mathbf{u}_{\mathcal{A}}$ are the information bits encoded by the polar code from $\mathcal{A}$, and there are K information bits. They represent the actual bits that you want to transmit, and are placed in reliable channels. Frozen bits are denoted by $\mathbf{u}_{\mathcal{A}^C}$, and are placed in unreliable channels. They are also fixed to known values (typically all zeros) and communicated to the decoder. Although frozen bits do not carry information, they are still needed to achieve channel capacity. The functions `select_information_bits` and `make_frozen_mask` in our code are responsible for selecting the proper bits.

III. How Polar Codes Work

A. Channel Combining and Encoding

The purpose of channel combining is to combine N copies of a B-DMC W recursively so it produces a new vector channel $W_N : \mathcal{X}^N \to \mathcal{Y}^N$, where N must be a power of 2 [6]. The construction of W_N is essential because its structure allows sub-channels to polarize into either highly reliable or highly unreliable channels.

To help simplify the explanation, we will consider the simplest non-trivial case where we are combining two independent channel uses of W. If we are given the two input bits $u_0, u_1 \in \{0,1\}$ from the message vector $\mathbf{u}$, the transmitted bits $x_0 = u_0 \oplus u_1$ and $x_1 = u_1$ are sent through two independent copies of W. Since the encoder rearranges the input bits prior to transmission, the resulting joint channel law is expressed in terms of (u_0, u_1):

$$W_2(y_0, y_1|u_0, u_1) = W(y_0|u_0 \oplus u_1)W(y_1|u_1).$$

For the general-case, the transmitted bits x can be expressed by:

$$x = uF$$

where F is Arıkan's kernel,

$$F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

For a general block length of $N = 2^n$, the combining operation is done recursively to eventually create W_N. The use of the XOR operation serves as the foundation for channel polarization, where synthesized sub-channels $W_N^{(i)}$ become highly reliable or highly unreliable as N grows.

Another key finding from Arıkan is that the same transformation used for channel combining can also be used for encoding [1]. The polar encoder maps an input vector u_0^{N-1} to the transmitted codeword

$$x = uG_N$$

where G_N corresponds to the generator matrix

$$G_N = F^{\otimes n}$$

where $\otimes n$ is the Kronecker power.

The entire transformation process is implemented via the `polar_encode` function in our Jupyter notebook. It is important to note that our implementation does not explicitly build G_N, since it is a large $N \times N$ matrix that would require an inefficient amount of computing power to create. Instead, we recursively called the `polar_encode` function to pass in x_0 and x_1 as parameters to perform the transformation.

B. Channel Splitting and Successive Cancellation Decoding

In order for the decoder to decide how it should interpret each input bit of $\mathbf{u}$, it uses the process channel splitting. Channel splitting results in N synthetic bit-channels, which are artificial, derived channels that describe the conditions under which SC decoding takes place. More specifically, they describe how reliably a decoder can estimate the bit u_i, and are denoted by:

$$W_N^{(i)} : \mathcal{X} \to \mathcal{Y}^N \times \mathcal{X}^{i-1}, 1 \le i \le N.$$

An important thing to note from this definition is that decisions for earlier bits impact later ones for $W_N^{(i)}$.

Successive cancellation (SC) decoding is the part in polar codes that uses the channel splitting mechanism. For each bit u_i obtained in the natural order, the decoder estimates

$$\hat{u}_i = \begin{cases} u_i, & \text{if } u_i \text{ is frozen,} \\ \arg\max\limits_{b\in\{0,1\}} W_N^{(i)}(y_0^{N-1}, \hat{u}_0^{i-1} \mid b), & \text{if } u_i \text{ is an information bit.} \end{cases}$$

Informally, this means that if u_i is a frozen bit, the decoder sets $\hat{u}_i = u_i$. If u_i is an information bit, the decoder sets $\hat{u}_i$ to 0 or 1 that is more consistent with what the channel output looked like.

In the `sc_decode` function of our Jupyter notebook, the function itself does not evaluate the synthetic channel $W_N^{(i)}$ directly. Instead, it calculates the log-likelihood ratios (LLRs) of $W_N^{(i)}$ [7] through the functions `f_min_sum` (fast approximation) or `f_exact` for upper-half LLRs, and `g_llr` for lower-half LLRs.

IV. Tests and Results

A. Deterministic Tests

Before any Monte Carlo tests were performed, a deterministic test was performed to check for the correctness of the polar code implementation. This test was completely noiseless to ensure that results could be reproduced, so `p_channel=0`. The test initially involved using a polar code with $N = 8$ and $K = 4$, and testing it on three random messages to see if the output would match the initial message, demonstrating proper encoding and decoding. The entire process was printed to help during debugging. Once these tests passed, we re-ran the test using $N = 16, 32, 64, 128$ and $K = N/2$ 50 times for each value of N to see if the results were still correct. When we saw that all the tests passed, we knew that the implementation was working correctly and we could proceed to Monte Carlo simulations.

B. Monte Carlo Simulations

Monte Carlo simulations were performed by running the code for 10,000 trials with randomly generated messages. These tests generally used the bit error rate (BER) and block error rate (BLER) to measure performance.

In the first simulation, we demonstrated one of the benefits of channel polarization, which is that BER decreases as the block length N increases. The polarization theorem states that a fraction of the synthetic channels converge to perfect channels, meaning that $Z(W_N^{(i)}) \to 0$. The remaining fraction converge to pure-noise channels, so $Z(W_N^{(i)}) \to 1$. Therefore, as N increases, the reliable channels become even better, and the unreliable channels become even worse, making it easier to distinguish between the two and transmit error-free information.

A constant noise value of 0.05 was used for this simulation. The plot generated from the simulation can be seen in figure 1. Although BER decreases throughout the plot, BLER does not at $N = 128$. This can be attributed to the noise present in the simulation.

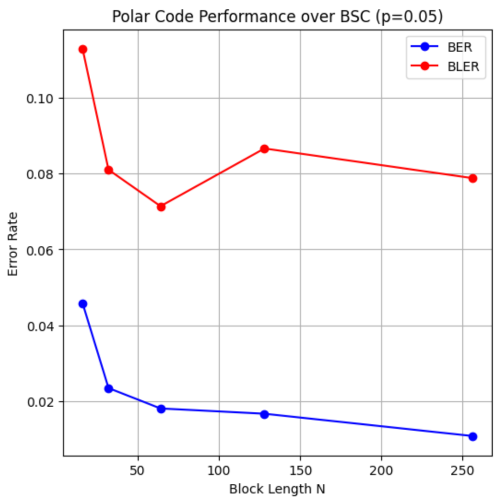

Fig. 1. Plotting BER AND BLER for increasing block lengths

The second simulation tested how increasing the noise effects the overall performance of polar codes. The values of N and K were fixed to 128 and 64 respectively for this test, while the noise (`p_channel`) was varied from values ranging from 0.01 to 0.20. As expected, increasing noise also increased the BER and BLER, with the BLER specifically reaching a maximum error rate of approximately 1.0 when the noise was 0.20. The plot generated from the simulation can be seen in figure 2 below.

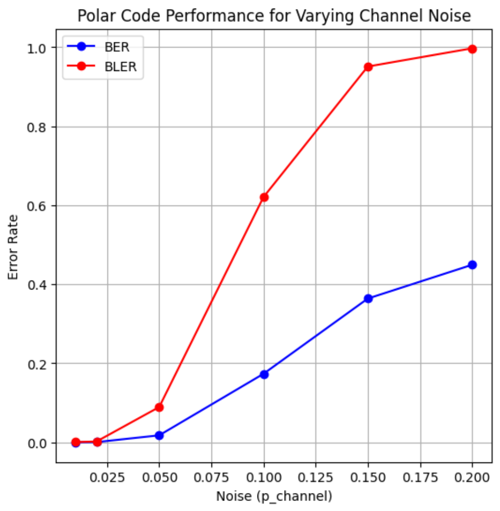

Fig. 2. Plotting BER and BLER for increasing channel noise

The third simulation tested how setting K equal to ratios of N, where N was fixed to 128 and the noise to 0.05. The K/N ratio is also known as the code rate. The results from the simulation can be observed in figure 3.

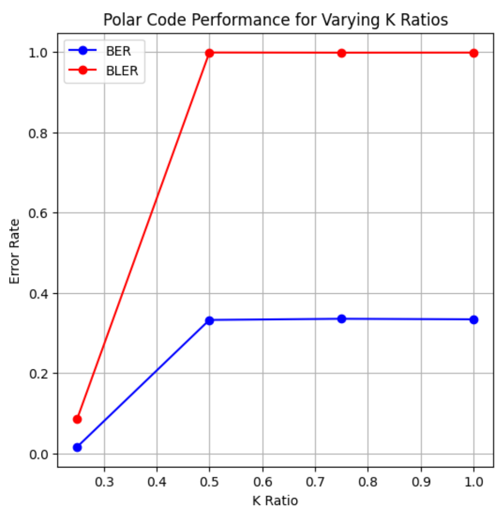

Fig. 3. Plotting BER and BLER for increasing code rates

As the K/N ratio increases, the code rate (ratio of information bits to transmitted bits) exceeds the channel capacity C, resulting in unreliable communication [1]. This forces the polar encoder to treat some synthetic channels with Bhattacharyya parameters Z_i close to 1 (unreliable) as reliable. As can be observed from the plot, the BER converges to around 0.33 due to the asymptotic error of the unreliable channels [1]. The BLER converges to 1, which is expected since the decoder will make at least 1 error for every block.

The final simulation demonstrates how the fraction of reliable channels increases as the block length increases. For this simulation, the noise (`p_channel`) was fixed to a value of 0.05, and the variable `Z_threshold` was fixed to 0.1, which determines how strict we are in determining a synthetic channel is reliable. A moderate threshold of 0.1 was chosen to visibly illustrate the growth of reliable channels for smaller values of N.

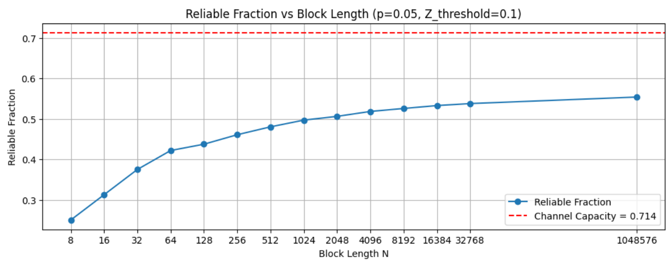

Fig. 4. Plotting block length vs fraction of reliable channels

From the plot in figure 4, it can be observed that as the block length N increases, the fraction of reliable channels also increases too. The channel capacity was plotted as a horizontal dashed line, which was calculated using `C = 1-binary_entropy(p)`. Similar to simulation 1, these results demonstrate how it is easier for the decoder to distinguish between reliable and unreliable channels, one of the key findings of channel polarization. Had the plot continued into values of N reaching the millions, the fraction of reliable channels would eventually converge to the channel capacity [1], but this was omitted since it would take too much computing power to execute.

V. Conclusion

Polar codes represent one of the most significant breakthroughs in error-correcting codes, as they are the first practical codes that can provably reach the Shannon capacity of B-DMCs. In this paper, we have described the theoretical foundations of polar codes such as channel polarization, channel combining and splitting, polar encoding, and successive cancellation (SC) decoding. Furthermore, we have implemented a Jupyter notebook using the Python programming language to test the effectiveness of polar codes through Monte Carlo simulations. This resulted in empirical evidence showing how they perform better at higher block lengths, confirming one of the key findings of channel polarization. We hope this paper serves as a useful introduction on the implementation of polar codes, and can be extended to other B-DMCs beyond binary symmetric channels (BSCs) for future use.

References

[1] E. Arikan, ”Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels,” in IEEE Transactions on Information Theory, vol. 55, no. 7, pp. 3051-3073, July 2009, doi: 10.1109/TIT.2009.2021379.

[2] C. Shannon, “A Mathematical Theory of Communication,” Bell System Technical Journal, vol. 27, no. 3, pp. 379–423, 1948, doi: https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

[3] R. Hamming, “Error Detecting and Error Correcting Codes,” Bell System Technical Journal, vol. 29, no. 2, pp. 147–160, Apr. 1950, doi: https://doi.org/10.1002/j.1538-7305.1950.tb00463.x.

[4] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1,” Proceedings of ICC ’93 - IEEE International Conference on Communications, vol. 2, Jun. 1993, doi: https://doi.org/10.1109/icc.1993.397441.

[5] T. Cover and J. Thomas, Elements of Information Theory, 2nd ed. Hoboken, N.J.: Wiley-Interscience, 2006.

[6] L. Zhang, “Polar Coding Notes: Channel Combining and Channel Splitting” Dsprelated.com, Oct. 19, 2018. https://dsprelated.com/showarticle/1201.php (accessed Dec. 09, 2025).

[7] I. Tal and A. Vardy, “How to Construct Polar Codes,” IEEE Transactions on Information Theory, vol. 59, no. 10, pp. 6562–6582, Oct. 2013, doi: https://doi.org/10.1109/tit.2013.2272694.