Sum Product Algorithm (SPA) for LPDC Codes

Simon, Hanzala and Baccha

Department of Iot and Wireless Technologies
Skolkovo Institute of Science and Technology
Moscow, Russia

Abstract- The Sum-Product Algorithm (SPA) which enables efficient decoding of Low-Density Parity-Check (LDPC) codes plays a significant role in the modern error correction techniques for digital communication systems. Theoretical groundwork, implementation, and performance study: Sum-Product Algorithm for Low Density Parity Check codes The SPA does this through the use of factor graphs and message passing to iterate over approaches that arrives at convergence with speed and accuracy enough to rival the optimal decoding output in various conditions, including channels operating with large levels of noise. SPA experiences significantly better iterative decoding performance across various channel models, and both message and iterative decoding complexities have been studied in detail in this paper, where its outstanding network throughput and reliable communication links have been discussed. The results of the simulation show that the algorithm can effectively decrease the bit error rates (BER) and improve the system reliability under different signal-to-noise ratio (SNR). The results highlight the importance of SPA in 5G and beyond communication systems, where low-latency and high-reliability requirements prevail.

1. Introduction

1.1 Overview of LDPC codes

An effective communication system in terms of capacity, speed, cost, lossless data transmission, and secure communication is essential for any improvements in the rapidly evolving fields of digital wireless communication and information technology. Since we have no control over the channel, changes in the losses caused by noise in the channel during wireless data transmission are not possible. In order to recover the original data and ensure a successful wireless transmission, an effective channel coding scheme is required. This scheme should be able to identify and fix any errors at the receiver that are brought on by noise in the wireless channel. However, as information technology advances and the amount of data in bits increases, a number of parameters must be improved, including the cost of implementation, the complexity of the structure for implementing such a logic of channel coding, and the speed at which data is encoded and decoded, or more accurately, computation time. Thus, LDPC code implementation can meet these many requirements for a significantly higher level of efficiency.

Since they provide a workable implementation that almost reaches the Shannon channel capacity of dependable transmission, LDPC codes are incredibly effective codes. According to the Shannon channel capacity rule, when we increase the block length, the code with a code rate close to the capacity number causes the error to go to zero when decoding using the greatest probability decoder. Random linear block codes that are encoded as polynomials of time can be used to meet this requirement. However, the problem of complicated computing techniques for encoding and decoding arises when we raise the block length for error near zero, which ultimately forces us to make a speed tradeoff. Thus, in this case, LDPC code successfully implements LDPC code at the Shannon limit with extended block lengths. It also offers the added benefits of faster and more accurate execution, as well as simpler algorithms.

1.2 Importance of Sum Product Algorithm.

The Product Algorithm is defined as an algorithm for computing marginals by sending messages between variables and functions in a factor graph, where function nodes send messages to connected variables and vice versa until each node has received messages from its neighbours. The algorithm terminates when the marginal probability of each node is computed based on incoming messages from all connected functions.

The SPA is one of the most important components of modern error-correcting code theory because it decodes LDPC codes very efficiently by using the inherent structure of the Tanner graph. Its iterative nature helps it approximate MAP decoding, which is computationally infeasible for large systems. With its exchange of probabilistic messages between variable and check nodes, SPA attains a very high decoding accuracy with computationally manageable complexity. Its application spans from wireless communications to data storage systems, making it a basic tool in achieving reliable data transmission over noisy channels.

1.3 Objective of the project

The main goal of this project is to implement the Sum-Product Algorithm to decode LDPC codes. Specifically, it will involve encoding an MNIST digit using given LDPC parity-check and generator matrices, transmission of the encoded data through a noisy channel with Gaussian noise, and application of the SPA for iterative decoding of the noisy signal. The project will visualize the evolution of the decoding process over 15 iterations, evaluate the performance of the decoding, and demonstrate the effectiveness of SPA in recovering corrupted data under realistic communication conditions.

2. Background

2.1 LDPC Code Basics

In LDPC code, the parity check sets can be expressed in terms of Sparse Parity Check Matrix. In a Sparse Parity Check Matrix which is having (n-k) × n dimension, the word "Sparse" means that the number of times '1's are very less than the number of times '0's. Normally for a large bit stream, the LDPC parity check matrix are of 1000×2000 dimensions. So, out of n×(n-k) entries, the number ones are very less than the number of zeros. There are three parameters that define the sparse parity check matrix which are (n, wc, wr). Here, n is coded length, wr is the number of ones in a row and wc is the number of ones in a column. For a matrix to be called low-density or sparse, the condition - wc and wr << n×(n-k) be satisfied. To satisfy this, the parity check matrix must be very large.Here, a 4×8 sparse parity check matrix H of (8,4,2) is shown.Each row of matrix H represents a check node and each column of matrix H represents variable nodes.

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

Figure 1: Sparse Parity Check Matrix

Variable nodes indicate the elements of the code word. Check nodes indicate the set of parity check conditions. Here, as per the H matrix illustrated above, wherever there is a 1 in the matrix, it denotes that there is a connection between check node and variable node.
Parity Check Matrix can be classified into two types.
1) Regular Parity Check Matrix: If the parity check matrix has uniform wr and wc(same no of ones in column and row) we call that a regular parity check matrix. This type makes decoding less complex.
2) Irregular Parity Check Matrix: If wc and wr are different for different columns and rows then it is called an Irregular parity check matrix.

2.2 Tanner Graph Representation

An error-correcting code's parity check matrix is represented by a bipartite graph called a Tanner graph. The (N–K)-by-N parity check matrix is denoted by H. N bit nodes, also known as variable nodes, are represented by circles in the Tanner graph. Squares are used to represent N–K check nodes. If a one appears in row I and column J of H, then there is an edge between bit node I and check node J. For instance

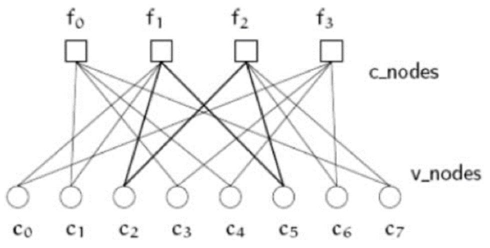

Figure 2: Tanner graph

Figure 2 is an example for such a tanner graph and represents the same code as the matrix from Figure 1. So, there will be n variable nodes and (n-k) check nodes. An edge or connection is made between the check node Ci and the variable node Vj if the element hij of matrix H is a 1. Check node Ci is connected to variable node Vj if the element hij of matrix H is a 1.Since the rows in H matrix refers to the check nodes and columns of matrix H refers to the variable nodes, we can see that there are 4 check nodes.

2.3 Theoretical Foundations of SPA

The set of VNs involved in the control equation i is defined as V(i) = {j : Hij =1}. The set of CNs involved in updating the VN j is designated by C(j)={i : Hij =1}. Moreover, V (i) \ j and C (j) \ i mean all the VNs of V (i) excluding node j, and all CNs of C (j) excluding node i, respectively.
And λj is the message coming from the logarithmic likelihood ratio of the message received yi determined by

$$\lambda_j = \ln\left(\frac{P(x_j = 0 \mid y_j)}{P(x_j = 1 \mid y_j)}\right)$$

.

αij and βij are the messages coming from CN i to VN j, and from VN j to CN i, respectively.

2.4 Decoding procedure:

Initialization:

The value λi is used to initialize βij for each i and j. The messages α and β are calculated and sent between VN and CN across the branch of the graph during each iteration, as in the following steps:

Check node update:

Using the βij values of all VN linked to the CN Ci to calculate the messages αij ,except the β message from Vij :

$$\alpha_{ij} = \prod_{j' \in V(i) \setminus j} sign(\beta_{ij'}) \times \varphi\left(\sum_{j' \in V(i) \setminus j} \varphi(|\beta_{ij'}|)\right)$$

$$\varphi(x) = -\log\left(\tanh\frac{|x|}{2}\right)$$

Variable node update:

Calculate βij using λj and α from all other CN's to VN j , except CN I:

$$\beta_{ij} = \lambda_j + \sum_{i' \in C(j) \setminus i} \alpha_{i'j}$$

Calculate the Syndrome and the termination condition:

For each column j the λj and α message from nearby CN's are added for update the wj, as following.

$$w_j = \lambda_j + \sum_{i' \in C(j)} \alpha_{i'j}$$

An estimated code $\hat{x}_i = \{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N\}$ is generated from the updated vector. Then xi is calculated by.

$$\hat{x}_i = \begin{cases} 1, & \text{if } w_i \le 0 \\ 0, & \text{if } w_i > 0 \end{cases}$$

If the condition H . xˆ T = 0 is satisfied, then the codeword is correct and the decoding ends. Otherwise, the procedure continues and the steps repeat until a valid codeword is found or I=Imax, this is when the decoding process is complete.

3. Problem Statement:

In today's digital communication systems, guaranteeing reliable data transfer over noisy channels is a crucial task. Low-Density Parity-Check (LDPC) codes have evolved as an effective error correcting scheme, capable of nearing Shannon's channel capacity. However, effective decoding algorithms are required to fully utilize the promise of LDPC codes. The Sum-Product Algorithm (SPA) plays an important part in doing this by iteratively refining probability estimates to accurately decode received signals. Despite its benefits, SPA implementation has hurdles in terms of convergence speed, computational complexity, and error floor behavior at low signal-to-noise ratios (SNR). This project seeks to address these issues by investigating the mathematical formulation, iterative message-passing mechanism, and performance analysis of SPA in the context of LDPC decoding.

3.1 Tools and Resources Used:

- **Programming Language:** Python
- **Dataset:** MNIST (for image data)
- **Libraries:** NumPy, SciPy, Matplotlib
- **Simulation Environment:** Jupyter Notebook

3.2 Steps of Implementation

The following procedures are involved in implementing the Sum-Product Algorithm (SPA) for LDPC code decoding:

Definition and Initialization of the Matrix:

The generating matrix (G) and parity-check matrix (H) can be defined programmatically or loaded from external files.
For every variable node, initialize the log-likelihood ratios (LLRs) using the channel observations that were received. The likelihood that a bit is 0 or 1 is represented by the LLR, which is based on the channel model (AWGN, for example).

Initialization of the Message:

Set up messages from variable nodes to check nodes (βij) and from check nodes to variable nodes (αij).
At the beginning, put $\beta ij = \lambda i$, where λi is the channel LLR for the i-th bit.

Message Passing Iteratively:

For every iteration: Update Check-to-Variable: Use the tanh rule to calculate the messages sent from check nodes to variable nodes:

Updates from variable nodes to check nodes are made by adding the initial LLR and adding the sum of incoming check node messages.

Soft and Hard Decisions:

After the iterative message-passing process:
Compute the final beliefs (soft decisions) by summing the LLR and incoming messages:
Perform hard decisions by applying a threshold:
The algorithm terminates if the syndromeis satisfied, indicating a valid codeword. If not, the process continues for a predefined number of iterations.

3.2.1 Loading and Preparing MNIST Data

The MNIST dataset, which is widely used for image classification and machine learning, provides the input data for evaluating the LDPC code implementation. The collection consists photos of handwritten digits (0-9), each measuring 28x28 pixels. LDPC encoding involves flattening pictures into 784-dimensional vectors.

3.2.2 Encoding the Message:

The first step in the transmission process is encoding the message, which makes sure that the information can be sent over a noisy communication channel in a fashion that allows for error correction. Encoding for Low-Density Parity-Check (LDPC) codes entails utilizing the generator matrix G to convert the input message into a codeword.

Message Representation: Usually, a binary vector u is used to represent the input message. The encoded codeword, or longer vector c, is created by multiplying this message vector by the generating matrix G. The message is mapped into a space that meets the parity-check requirements specified by the LDPC code thanks to the encoding.

Encoding Procedure: To create the encoded codeword, parity-check bits are added to the message bits. The original information bits and the extra parity bits needed for error correction are both included in the codeword c, which has a length of length N, where N is the block size. The communication channel will be used to send this codeword.

Transmission of the Codeword: Next, the channel is used to send the encoded message c. It is crucial to remember that although the encoding step aids in the creation of a codeword appropriate for error correction, the decoding procedure subsequently recovers the original message from the received signal using methods like the Sum-Product Algorithm (SPA).

3.2.3 Modulation:

After encoding, the binary codeword *c* is modulated into a signal that may be transmitted via a physical communication channel. Binary Phase Shift Keying (BPSK) is a common modulation method used in this step.

BPSK Modulation is a phase modulation technique that assigns each bit to one of two unique phases. In BPSK, a binary 0 is mapped to +1 (a phase of 0°), while a binary 1 is mapped to −1 (a phase of 180°). This is the most basic kind of phase modulation, which makes it resistant to noise and interference.

3.2.4 Adding White Gaussian Noise:

After generating the BPSK modulated signal, the following step is to simulate the effects of a noisy channel. Real-world communication systems are prone to noise, which can distort the received signal. This noise is typically modeled using additive white Gaussian noise (AWGN).

The AWGN channel is a common model that corrupts transmitted signals with Gaussian noise with zero mean and constant variance. This noise is added to the modulated signal, resulting in a noisy reception of the broadcast signal. Received Signal: The received Signal is presented as

$$r_i = s_i + n_i$$

3.2.5 Decoding Using SPA:

After the signal has been transmitted over the noisy channel and received with additional Gaussian noise, the next step is decoding. The Sum-Product Algorithm (SPA) iteratively decodes the received signal to retrieve the original transmission message.

The initial stage in SPA decoding is to set the log-likelihood ratios (LLRs) depending on the received signal. The LLRs are calculated for each bit using the received signal and noise characteristics. These initial LLRs serve as the foundation for the iterative message-passing mechanism.

The SPA method sends messages between variable nodes and check nodes in the Tanner graph. Initially, transmissions from variable nodes to check nodes are based on the received LLRs. The check nodes then iteratively improve these messages based on the parity-check restrictions. This procedure continues for multiple iterations, with the messages modified at each stage.

The check node algorithm uses the tanh rule to compute messages sent to variable nodes. This phase ensures that the messages carry the parity-check restrictions throughout the graph.

Variable nodes update messages to check nodes by adding the received LLR and the sum of incoming messages.

Belief Calculation: After multiple cycles, the final beliefs (or soft judgments) for each bit are determined by adding the LLRs and incoming messages. The program then makes a difficult decision by using a threshold to determine if each bit is 0 or 1.

The decoding procedure proceeds until the syndrome condition is met, indicating that a valid codeword has been discovered. If this condition is reached prior to reaching the maximum number of iterations, the algorithm ends. If not, the decoding procedure will continue until either a valid codeword is discovered or the maximum iteration limit is met.

3.4 Mathematically:

The set of VNs involved in the control equation i is defined as V(i) = {j : Hij =1}.
The set of CNs involved in updating the VN j is designated by C(j)={i : Hij =1}.
Moreover, V (i) \ j and C (j) \ i mean all the VNs of V (i) excluding node j, and all CNs of C (j) excluding node i, respectively.
And λj is the message coming from the logarithmic likelihood ratio of the message received yi determined by:

$$\lambda_j = \ln\left(\frac{P(x_j = 0|y_j)}{P(x_j = 1|y_j)}\right)$$

αij and βij are the messages coming from CN i to VN j, and from VN j to CN i, respectively.

The steps of the SPA decoding algorithm are:

1. Initialization:
The value λi is used to initialize βij for each i and j. The messages α and β are calculated and sent between VN and CN across the branch of the graph during each iteration, as in the following steps:
2. Check Node Update:

For each check node, update the messages αij sent to variable nodes by computing:
Where the φ(x) function is defined as:
This step ensures that messages reflect the parity constraints across neighboring variable nodes, propagating constraints iteratively.

$$\alpha_{ij} = \prod_{j' \in V(i)\backslash j} sign(\beta_{ij'}) \times \varphi\left(\sum_{j' \in V(i)\backslash j} \varphi(|\beta_{ij'}|) \right)$$

$$\varphi(x) = -\log\left(\tanh\frac{|x|}{2} \right)$$

3. Variable Node Update:

At each variable node, compute updated messages βij to check nodes using the received channel LLR λi and incoming messages α from connected check nodes:
Where C(j) represents all check nodes connected to the variable node j, excluding the one under consideration.

$$\beta_{ij} = \lambda_j + \sum_{i' \in C(j)\backslash i} \alpha_{i'j}$$

4. Syndrome and Termination Check:

Calculate the updated belief for each variable node by summing the LLR λj and all incoming messages:
Perform hard decision decoding to estimate the codeword bits:
The algorithm terminates if the syndrome H is satisfied. Otherwise, the procedure continues iteratively until a valid codeword is found or the maximum iteration limit is reached.

$$w_j = \lambda_j + \sum_{i' \in C(j)} \alpha_{i'j}$$

4. SPA IN THE LOG-LIKELIHOOD DOMAIN

A binary (N, K) LDPC code [1, 2] is a linear block code described by a sparse M x N parity-check matrix H, i.e., H has a low density of 1s. The parity-check matrix H can be viewed as a bipartite graph with two kinds of nodes: N symbol nodes corresponding to the encoded symbols, and M parity- check nodes corresponding to the parity checks represented by the rows of the matrix H. The connectivity of the bipartite graph is such that the parity-check matrix H is its incidence matrix. For regular LDPC codes, each symbol is connected to d. parity-check nodes and each parity-check node is connected to de symbol nodes.

Following a notation similar to [2, 5], let M(n) denote the set of check nodes connected to symbol node n, i.e., the po- sitions of 1s in the n-th column of the parity-check matrix H, and let N(m) denote the set of symbol nodes that partic- ipate in the m-th parity-check equation, i.e., the positions of Is in the m-th row of H. Furthermore, N(m)\n represents the set N(m), excluding the n-th symbol node, and similarly, **M(n)\m** represents the set M(n), excluding the mmm-th check node.

In addition, qn→m(x), x∈{0,1}, denotes the message that symbol node n sends to check node m, indicating the probability of symbol n being 0 or 1, based on all the checks involving n except m. Similarly, rm→n(x), x∈{0,1}, denotes the message that the m-th check node sends to the n-th symbol node, indicating the probability of symbol nnn being 0 or 1, based on all the symbols checked by mmm except nnn. Finally, y=[y1,y2,…,yN] denotes the received word corresponding to the transmitted codeword u=[u1,u2,…,uN]. The LLR of a binary-valued random variable U is defined as:

$$L(U) \overset{def}{=} \log\frac{P(U=0)}{P(U=1)},$$

Channel LLR for a received bit yn

$$L(y_n) = \log\frac{\Pr(y_n \mid u_n = 0)}{\Pr(y_n \mid u_n = 1)}$$

Messages are passed between nodes in LLR form, reducing operations to additions and hyperbolic functions.

Steps of the SPA in the LLR Domain:
The SPA alternates between check node updates and symbol node updates along the edges of the bipartite graph.

- Initialization: For each symbol node n, compute the channel LLR based on the received value yn.

$$q_{n\to m} = L(y_n), \quad \forall m \in \mathcal{M}(n)$$

This is the initial message sent from symbol nodes to check nodes.

- Iterative Message Passing:

For each edge (m,n), compute the outgoing message from check node mmm to symbol node n:

$$r_{m\to n} = 2\tanh^{-1}\left(\prod_{n' \in \mathcal{N}(m)\backslash n} \tanh\frac{q_{n'\to m}}{2} \right)$$

1. The **product** term represents the parity constraint at the check node, ensuring that the sum of connected bits satisfies the parity-check equation.
2. The tanh function transforms LLRs to probabilities.
3. The tanh-1 ensures the message is returned to the LLR domain.

Symbol Node Update:

For each edge (n,m), compute the outgoing message from symbol node n to check node m:

$$q_{n\to m} = L(y_n) + \sum_{m' \in \mathcal{M}(n)\backslash m} r_{m'\to n}$$

L(yn) represents the likelihood of the bit based on the channel observation. The **sum** term aggregates information from all check nodes except mmm, ensuring consistency across the graph.

A Posteriori LLR Calculation

After a fixed number of iterations, compute the a posteriori LLR for each symbol node:

$$L(u_n) = L(y_n) + \sum_{m \in \mathcal{M}(n)} r_{m \to n}$$

$$\hat{u}_n = \begin{cases} 0 & \text{if } L(u_n) > 0 \\ 1 & \text{otherwise.} \end{cases}$$

Covergence Criteria:

The SPA stops under one of the following conditions:

1. All parity-check equations are satisfied

$$\mathbf{H} \cdot \mathbf{u}^T = 0$$

where u is the decoded codeword. A maximum number of iterations is reached.

4. Experimental Setup:

To implement and evaluate the Sum-Product Algorithm (SPA) for decoding Low-Density Parity-Check (LDPC) codes, applied to an MNIST digit corrupted by Gaussian noise. The goal is to reconstruct the original digit through iterative decoding.

Image Size: 28x28 (MNIST standard)

Codeword Length: 256

Message Length: 128

SNR: 2.5 dB

Iterations: 15

4.1. Observations across Iterations

Iteration 1-5:

The initial iterations show significant noise, with only a faint outline of the digit visible. Most parity-check constraints are unsatisfied, leading to incomplete error correction. The reconstructed image is still far from the original.

Iteration 6-10:Gradual refinement of the digit is observed. Noise begins to diminish, and the overall shape of the digit becomes clearer. By iteration 10, approximately 70-80% of the image pixels match the original digit. Iteration 11-15: Near-complete reconstruction occurs, with most of the noise removed. Iterations 12-15 show diminishing returns, as the algorithm converges and the digit stabilizes. The final image closely resembles the original MNIST digit, demonstrating the effectiveness of SPA in handling noisy data.

4.2 Bit Error rate trend

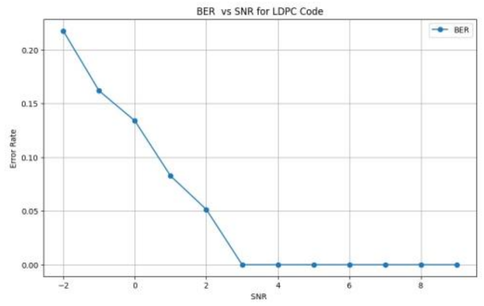

Fig 3. BER vs SNR

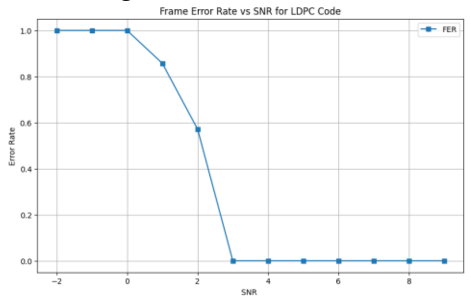

Fig4. FER vs SNR

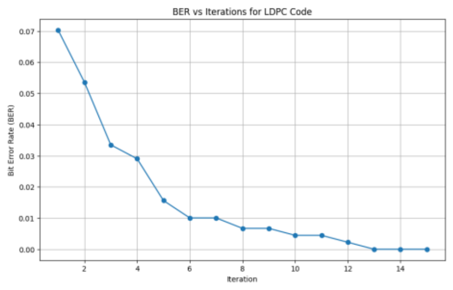

Fig 5. BER vs Iterations

4.3 Visual Evolution of Decoded Images

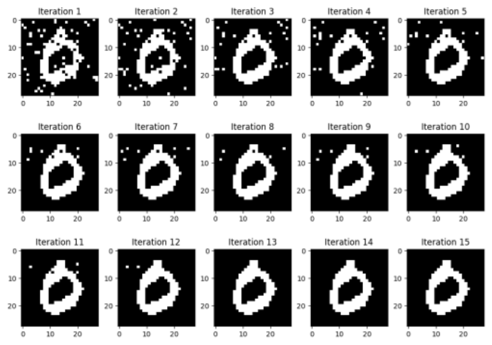

Fig 6.

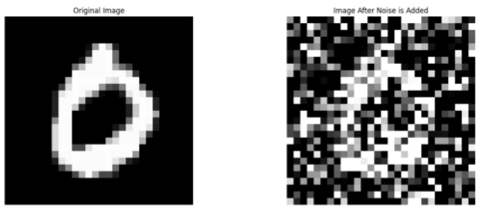

Fig 7.

4.4 Impact of SNR on Decoding Performance

Error correction is faster at higher SNR values but requires more iterations at lower SNR levels. Convergence typically occurs between 12-15 iterations for SNR values around 2.5 dB. Residual noise may persist for very noisy inputs, but the primary features of the digit are generally preserved.

4. Challenges and Solution:

Numerical Stability

Issue: Overow in Hyperbolic Tangent (tanh) and Inverse Hyperbolic Tangent (arctanh) Functions

The SPA algorithm iteratively passes messages between the variable nodes and check nodes, which are based on the tanh and arctanh functions. However, these mathematical functions can face a problem with numerical instability for the following reasons:

1. Range of tanh(x):

The function tanh(x) maps real numbers to the range (−1,1). For large values of x, tanh(x) asymptotically approaches −1 or 1. In the course of SPA iterations, messages
between nodes can increase considerably in magnitude and lead to values of tanh(x) that are very close to −1 or 1.

2. Behavior of arctanh(x):

The arctanh(x) is dened only within x∈(−1,1). In addition, minor numerical errors leading to very slight overruns of the input to \arctanh(x), such as x=1.00001, usually result in undened values.

3. Accumulation of Errors:

For every tanh value, product, and division involved in this SPA, an iterative buildup of small numerical inaccuracy can quickly push them into invalid ranges.

Solution: Clipping Values to a Safe Range

To address this issue, the algorithm implements clipping, a technique that ensures all values remain
within a numerically safe range.

1. Clipping Mechanism:

Before applying arctanh(x), the values of x are clipped to a slightly narrower range:
x∈[−1+ϵ,1−ϵ]where ϵ is a small positive constant (e.g., 10−16).
In the code, this is achieved using: values_tanh = np.clip(values_tanh, -1 + 1e-16, 1 - 1e-16)

2. Why Use Clipping?

Undened Behavior Avoidance: This will make sure that all the inputs to arctanh(x) are well-dened and never result in either inf or NaN.
It provides numerical stability: the selected range is close enough to −1 and 1 to retain accuracy while being able to prevent instability.
Prevents Overow: Reduces the risk of computational overows during iterative updates.

3. Impact on the Algorithm:

Clipping does not signicantly affect the accuracy of SPA decoding because the values clipped are usually so close to −1 or 1 that their contribution to the nal result can be considered negligible. It ensures the stability of the algorithm across iterations, even for difcult noise conditions or large parity-check matrices.

Example of Clipping in Action

Suppose the SPA calculates the following tanh values during an iteration: values_tanh = [-1.000001, -0.95, 0.98, 1.000001] Without clipping, the \arctanh function would fail for −1.000001 and 1.000001.

After clipping:

values_tanh = [-1 + 1e-16, -0.95, 0.98, 1 - 1e-16] These adjusted values are now safe for further processing.

5. Conclusion:

This project presented SPA for the decoding of LDPC codes and showed, in general, that one can recover corrupted data indeed. By iteratively enhancing log-likelihood ratios within the Tanner graph structure, SPA gave a correct re construction of a noisy MNIST digit. It noted the critical challenges faced in its implementation, including numerical stability, noise handling, and several others, with their subsequent robust solutions such as clipping of values and judicious choice of SNR, among others. This implementation underlines the importance of SPA in modern error correction and its potential for wider applications in communication and data processing systems.

References

R. G. Gallager, “Low Density Parity Check Codes,” Sc.D. thesis, Mass.
Inst. Tech., Cambridge; September, 1960.
C. E. Shannon, “A mathematical theory of communication,” Bell System
Technical Journal, vol. 27, pp. 379–423, 1948.
Tu, Zongjie, and Shiyong Zhang. "Overview of LDPC codes." 7th IEEE
International Conference on Computer and Information Technology (CIT
2007). IEEE, 2007..
Sybis, Michal, et al. "Channel coding for ultra-reliable low-latency
communication in 5G systems." 2016 IEEE 84th vehicular technology
conference (VTC-Fall). IEEE, 2016
Kienle, Frank, Torben Brack, and Norbert Wehn. "A synthesizable IP
core for DVB-S2 LDPC code decoding." Design, Automation and Test in
Europe. IEEE, 2005.M.