# Iterative Decoding of Low Density Parity Check Codes

**Saptarashmi Bandyopadhyay**
**(Examination Roll - 510514006 )**
**Eighth Semester 2017-2018**
**B.Tech. in Computer Science and Engineering**
**Computer Science and Technology Department**
**Indian Institute of Engineering Science & Technology, Shibpur**

Under the Esteemed Guidance of
Dr. Abhik Mukherjee

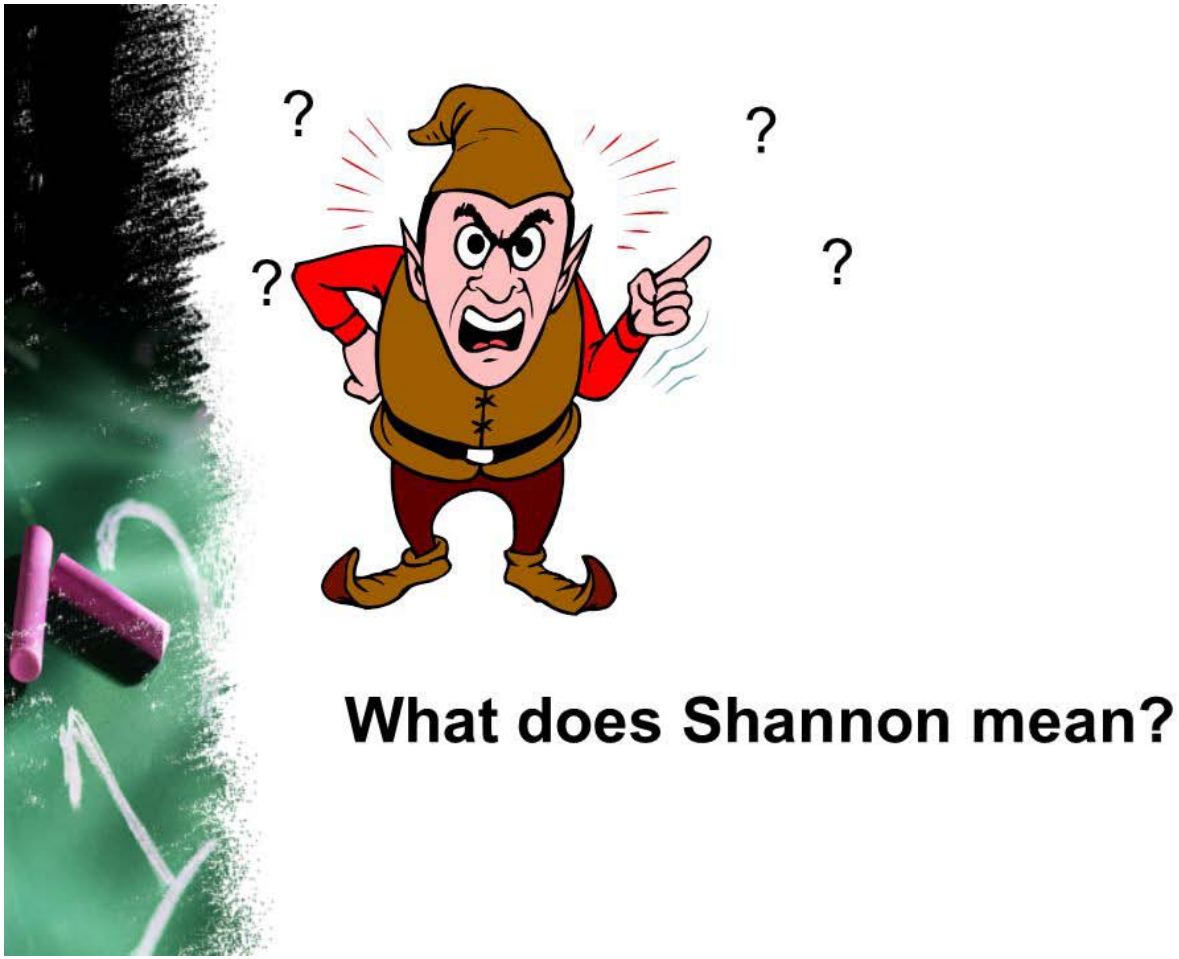
# Outline

- **Low density parity check codes**
- **Tanner Graph Representation**
- **Parameters of LDPC Code**
- **Encoding using BPSK Modulation and AWGN noise**
- **Bit Flipping Algorithm**
- **-Multi-Bit Flipping Hard Decision decoding**
- **Message Passing Algorithm**
- **Message Passing with Bit Flipping**
- **Error Correction**
- **Experimentation**
- **Optimistic Noise selection in AWGN Channel**

# Outline

- **Probabilistic Decoding Algorithm**
- **Log likelihood decoding algorithm**
- **Why Not minimum distance based decoding?**
- **Deep Learning in LDPC Decoding**
- **Neural network algorithms**
  - **Sequential training mechanism**
  - **Aggregate training mechanism**
- **Limitation of Neural Network approach**
- **Future Work**
- **References**

# Importance of Channel Capacity from Shannon's theorem



# Low-density parity check codes

- A low-density parity-check code, or LDPC[1] code, is defined by a matrix $H \varepsilon \{0,1\}_{mxn}$.
- H is sparse and has O(n) nonzero elements.
- The set of code words C is defined by

  $$C = \{ x \varepsilon \{0,1\}^n \mid H x^T = 0^T \};$$

  where 0 is the zero vector and we assume that all vectors are row vectors.
- The matrix H is referred to as the parity-check matrix, since it requires that a code word x have even parity in the so-called parity-check equations.

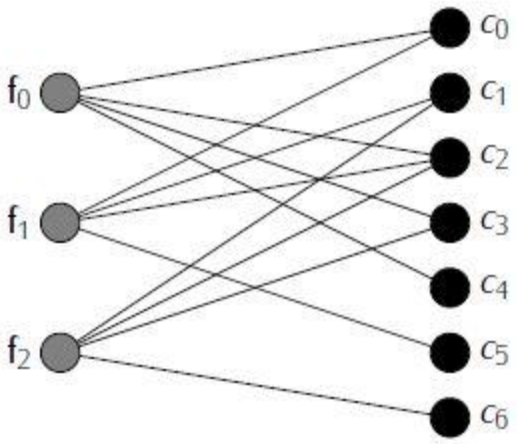
# Low-density parity check codes

- $s_a = \sum_{i=1}^{n} H_{ai} x_i = 0$; for all a = 1,2, …, m.
- Each parity-check equation is often called a checksum or check, and the vector s of sums $s_a$ is called the syndrome.
- If a check has even parity we say that the check is satisfied and otherwise we say that the check is unsatisfied.
- The term ‘’low density’’ signifies a sparse parity check matrix in terms of the number of 1s present in the matrix.
- Due to this sparseness, LDPC codes possess great advantages in terms of efficient decoder implementation and storage and can approach the Shannon Limit .

# Tanner Graph Representation

- The graph representation of a parity-check matrix is more commonly referred to as its Tanner graph.
- The Tanner graph is a bipartite graph with two sets of nodes.
  - check nodes, corresponding to the rows of H.
  - variable nodes, each corresponding to a column of H.
- There is an edge between a check node *a* and a variable node *i* if and only if $H_{ai} = 1$.

| $[H]_{(3\times7)}$ | $c_0$ | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ | $c_6$ |
|---|---|---|---|---|---|---|---|
| $f_0$ | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| $f_1$ | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| $f_2$ | 0 | 1 | 1 | 1 | 0 | 0 | 1 |




$$f_0 = c_0 + c_2 + c_3 + c_4 = 0$$
$$f_1 = c_0 + c_1 + c_2 + c_5 = 0$$
$$f_2 = c_1 + c_2 + c_3 + c_6 = 0$$

# Parameters of the LDPC code considered

- K(size of input message)=1000
- N(size of received output message with redundancy)=2000
- Number of frames:10
- Number of iterations:10
- Parity check Matrix of Dimension 1000*2000.
- In the parity check matrix
  - The number of ones in the row = 6
  - The number of ones in the column = 3

# LDPC toolkit

- Open source LDPC framework [3] in MATLAB.
- It has been adapted and coded according to experiments.
- The encoding framework involving BPSK modulation and addition of AWGN (Additive White Gaussian Noise) from the framework.

# BPSK for Encoding

- BPSK (Binary Phase Shift Keying) Modulation used in the toolkit.
- a digital modulation scheme that conveys data by changing, or modulating, two different phases of a reference signal.

- bpskMod = 2*u - 1; where u is the input message.

# Addition of AWGN

- Additive : the received signal is equal to the transmitted signal plus noise.

$$r = v + e$$

- White noise implies uniform power across the whole frequency band.
- The probability distribution of the noise samples is Gaussian.
- N0 = 1/(exp(EbN0(i)*log(10)/10));
  - Where EbN0 = [0, 0.5 ,1 ,1.5] in the experiment.

# Bit-Flipping Algorithm

- This algorithm attempts to correct a single bit in every iteration.
- If multiple bits are allowed to flip in one iteration, then the algorithm is called a Multi-Bit-Flipping Algorithm.
- In every iteration of this algorithm the code word bit(s) with the highest number of a metric are flipped.

# Bit-Flipping Algorithm

- Metric is characterized by the type of algorithm: Hard-Decision and Soft-Decision.
  - Hard decision metric = the number of unsatisfied parity check equations.
  - Soft decision metric = both the number of unsatisfied and satisfied parity check sums, since reliability information is present.
- An algebraic decoding algorithm in this context can only correct up to its random error correcting capability.
- But due to the sparse nature of H, the BF algorithm may correct error patterns whose number of errors exceeds the error correcting capability of the code.

# Multi Bit-Flipping Hard Decision Decoding

- 1. initialize iteration=0, Failure[i]=0 for the i-th bit, i=0,1,..,n-1 and preset a maximum number of iterations $i_{max}$.
- 2. Compute the syndrome of the received vector r. If the syndrome is 0, i.e., all the parity check equations are satisfied, the decoding is stopped and r is declared as the code word.
- 3. set iteration=iteration+1. Compute the number of failed parity check equations for each bit and store them in Failure[i] for the i-th bit, i=0,1,..,n-1
- 4. Construct a set S = {j: Failure[j]>= Failure[j'] for all j=0,1,..n-1}, i.e., compute the set S of bits for which the Failure[j] is the largest
- 5. Flip the bit(s) in S
- 6. Repeat steps 2 to 5 until the syndrome is 0 or iteration <= $i_{max}$

# Error Correction in Bit Flipping Based Decoding

- The input message consists of all zeros as any error in the decoded message can be calculated from the relative number of ones compared(x) to the entire error message (w).
- Eg.
- Number of non zeros after decoding for Noise point 1 (0.5dB): 203
- The bit error rate should be 203/2000.
- The bit error rate is 0.1015000.

# Standard Deviation and Mean for Bit flipping

| Noise (in dB) | 0.5 | 1 | 1.5 | 2 | 2.5 | 3 |
|---|---|---|---|---|---|---|
| Standard Deviation | 0 | 0.5896917 | 0.2138452 | 0.2962605 | 0.1748706 | 0.1063528 |
| Mean | 1.323500 | 0.537025 0 | 0.230983 3 | 0.259937 5 | 0.179120 0 | 0.132750 0 |

Implication that much of the data over individual noise is spread around the mean.

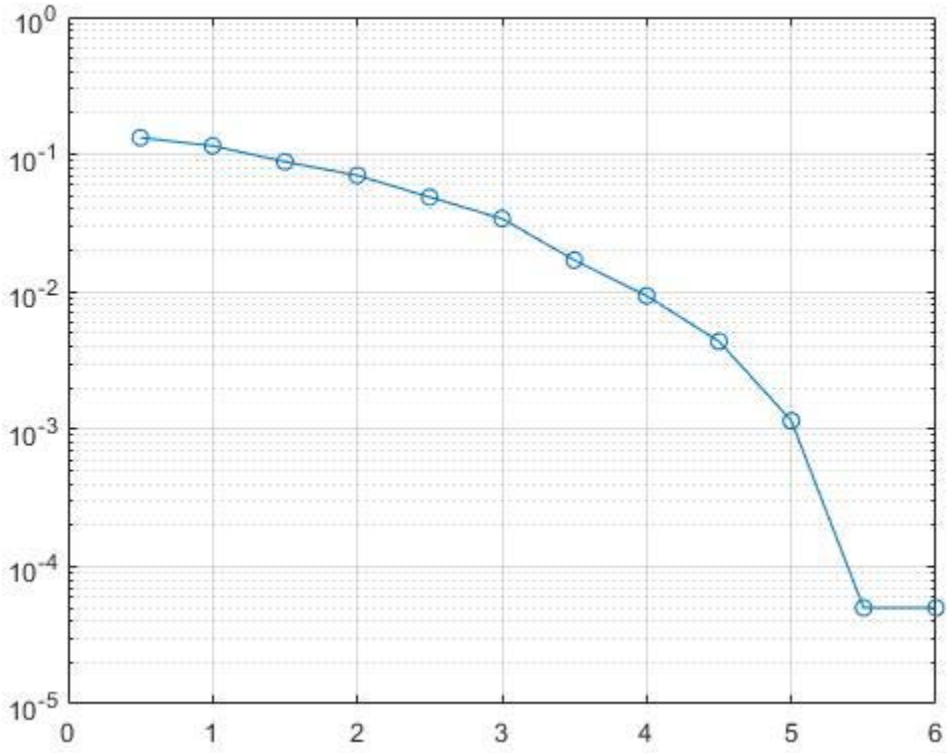
- The overall standard deviation : 0.04757824
- The overall mean : 0.04346250
- 12 Test cases are considered and then averaged over.

# Standard Deviation and Mean for Bit flipping

| Noise (in dB) | 3.5 | 4 | 4.5 | 5 | 5.5 | 6 |
|---|---|---|---|---|---|---|
| Standard Deviation | 0.0481639 6 | 0.0399786 0 | 0.0428141 5 | 0.0464497 8 | 0.0477516 1 | 0.0475410 7 |
| Mean | 0.0942285 7 | 0.0750125 0 | 0.0621611 1 | 0.0531800 0 | 0.0474500 0 | 0.0435000 0 |

# Bit error rate to noise in Bit flipping

^
|
|
|
Y AXIS
: BER




X AXIS : Eb/N0 ---------->

# Message Passing (MP) Algorithm

- Messages are passed between message nodes and check nodes along the edges of the Tanner graph. [3]
- The information that is sent from a node $v_i$ to $v_j$ in an MP algorithm should not get influenced by the node $v_j$ i.e., in other words, the data that is being sent to a node must depend only on the other nodes.
- This type of message is called an “extrinsic message” or “extrinsic information”.

# Message Passing Algorithm

- The generic message passing structure is as follows :
- 1.Initialize the decoder by sending the received values of the message nodes to check nodes.
- 2.In each check node, an extrinsic message for each neighbor variable node is calculated and sent.
- 3.Each message node calculates a new value for itself depending on the previous received values and the information from the check nodes. The output is then checked by all the parity check equations and if all of them agree, decoding stops. Otherwise, an extrinsic message for each neighbor check node is calculated and sent.
- 4.Repeat steps 2 and 3 until a maximum number of iterations is reached.

# Error Correction in Message Passing

- Example.
- Number of non zeros after decoding for Noise point 1(0.5dB): 1059
- The bit error rate should be 1059/2000.
- The bit error rate is 0.5295000

# Standard Deviation and Mean for Message passing

| Noise (in dB) | 0.5 | 1 | 1.5 | 2 | 2.5 | 3 |
|---|---|---|---|---|---|---|
| Standard Deviation | 0 | 7.227692 | 2.810648 | 2.436291 | 2.193719 | 1.929314 |
| Mean | 3.323500 | 3.048000 | 2.181550 | 1.771588 | 1.531790 | 1.337342 |

Implication that much of the data over individual noise is spread around the mean.

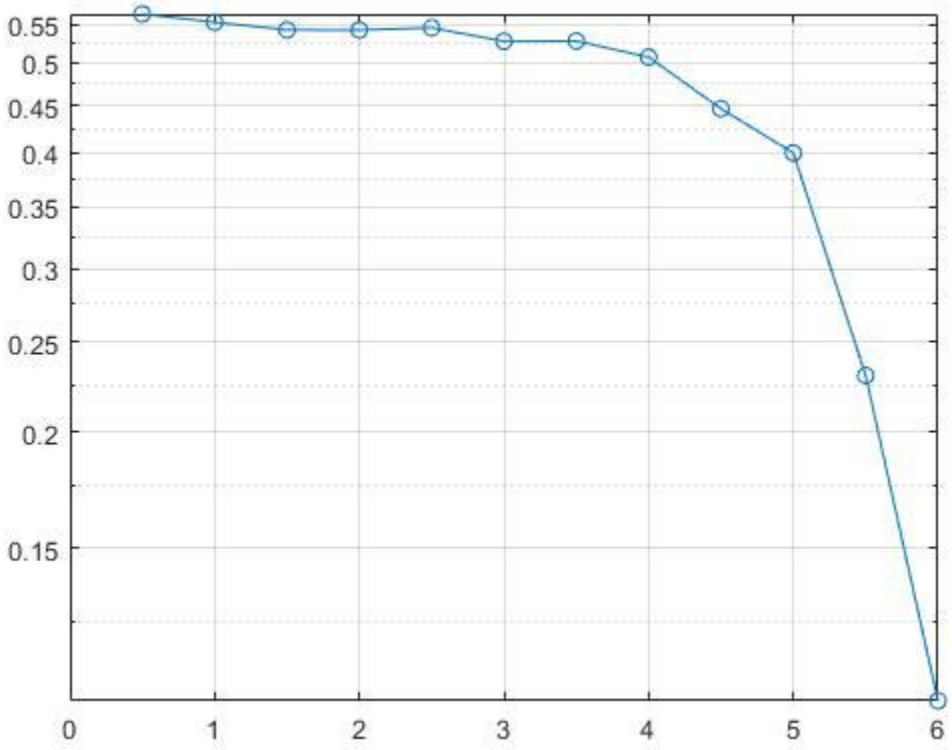
- The overall standard deviation : 0.1465706
- The overall mean : 0.4574958
- 12 Test cases are considered and then averaged over.

# Standard Deviation and Mean for Message passing

| Noise (in dB) | 3.5 | 4 | 4.5 | 5 | 5.5 | 6 |
|---|---|---|---|---|---|---|
| Standard Deviation | 1.788746 | 1.601317 | 1.307971 | 1.097616 | 0.5405744 | 0.1815345 |
| Mean | 1.222086 | 1.109556 | 0.9748556 | 0.8755450 | 0.6780227 | 0.5344458 |

# Bit error rate to noise in Message passing

^
|
|
|
Y AXIS
: BER




X AXIS : EbNo ----------->

# Message Passing with Bit-Flipping

- 1.Initialize the decoder by sending the received values of the message nodes to check nodes.
- **2. The message nodes do bit-flipping based on the number of unsatisfied parity check equations. (Hard Decision Bit Flipping)**
- 3.In each check node, an extrinsic message for each neighbor variable node is calculated and sent to the message nodes.
- 4.Each message node calculates a new value for itself depending on the ~~previous received values~~ **bit-flipped value** and the information from the check nodes. The output is then checked by all the parity check equations and if all of them agree, decoding stops. Otherwise, an extrinsic message for each neighbor check node is calculated and sent.
- 5.Repeat steps 2, 3 and 4 until a maximum number of iterations is reached.

# Error Correction for message passing with bit flipping

- Example.
- Number of non zeros after decoding for Noise point 1(0.5 dB): 183
- The bit error rate should be 183/2000.
- The bit error rate is 0.09100000.

# Standard Deviation and Mean for message passing with bit flipping

| Noise (in dB) | 0.5 | 1 | 1.5 | 2 | 2.5 | 3 |
|---|---|---|---|---|---|---|
| Standard Deviation | 0 | 0.9449068 | 0.6314297 | 0.4730536 | 0.3372613 | 0.2427406 |
| Mean | 1.712000 | 0.839350 0 | 0.525483 3 | 0.385475 0 | 0.289670 0 | 0.227583 3 |

Implication that much of the data over individual noise is spread around the mean.

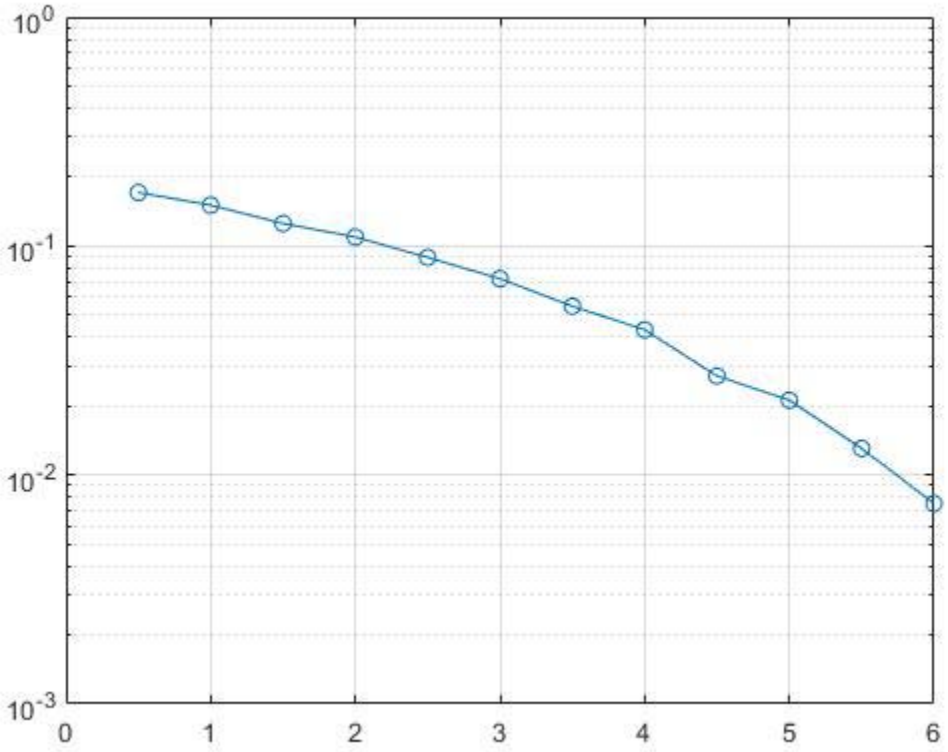
- The overall standard deviation : 0.05554713
- The overall mean : 0.07369583
- 12 Test cases are considered and then averaged over.

# Standard Deviation and Mean for message passing with bit flipping

| Noise (in dB) | 3.5 | 4 | 4.5 | 5 | 5.5 | 6 |
|---|---|---|---|---|---|---|
| Standard Deviation | 0.1645211 | 0.1196909 | 0.0705511 5 | 0.0597802 7 | 0.0510306 3 | 0.0515050 3 |
| Mean | 0.1804929 | 0.1503750 | 0.0120672 2 | 0.1054600 | 0.0903909 | 0.0793208 3 |

# Bit error rate to noise for message passing with bit flipping

^
|
|
|
Y AXIS
: BER



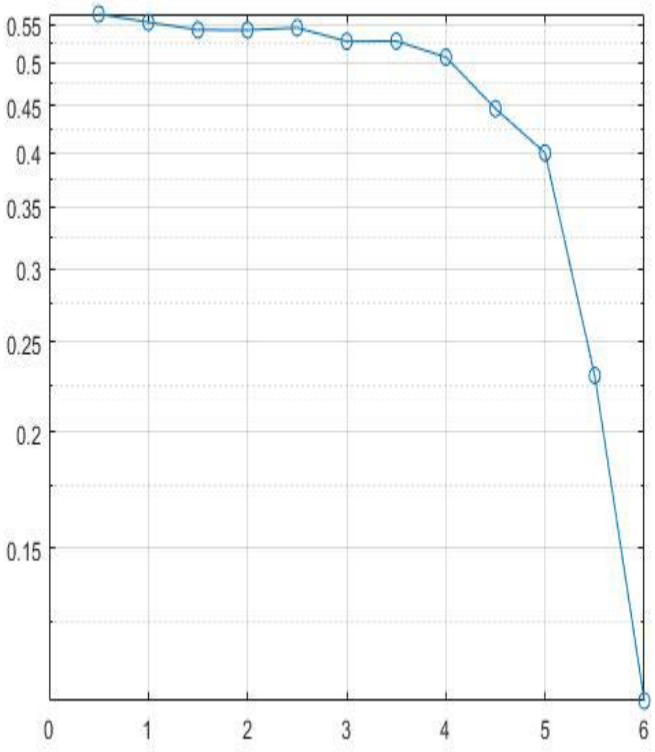

X AXIS : Eb/N0 ---------->

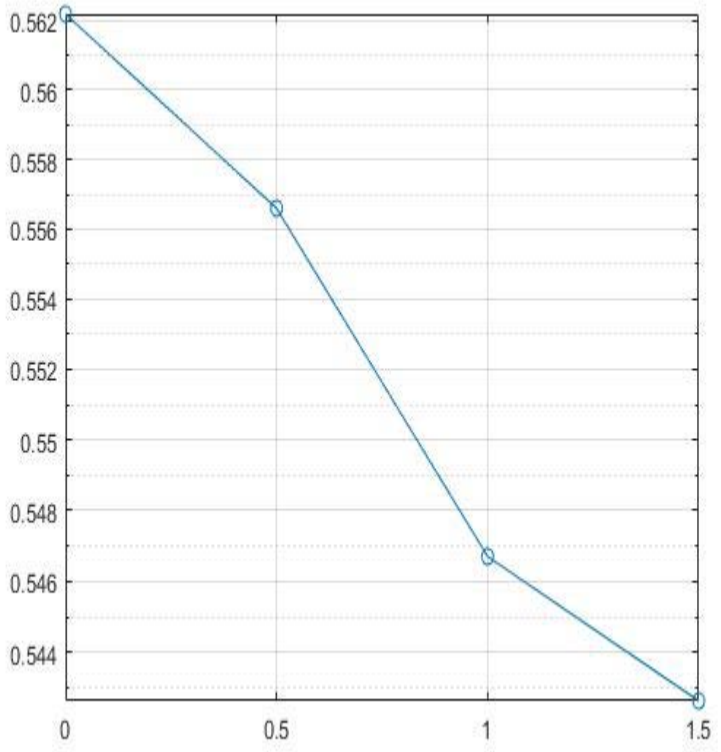
# Selection of optimistic noise in AWGN channel

- The selection of optimistic noise is based on Shannon Hartley equation of
  - $C=B*\log_2 (1+(S/N))$
    - where C is maximum capacity of the channel in bits/s.
    - B is bandwidth of the channel in hertz.
    - S is signal power in Watts.
    - N is noise power in Watts.
- For binary digits across AWGN (Additive White Gaussian Noise Channel) at transmission rate=channel capacity
- $C/B=\log_2 (1+(E_b/N_0)(C/B))$ or, $E_b = (B/C)(2^{(C/B)}-1)=(2^x-1)/x$ where x is C/B.
- Optimistic noise can be considered to be those noise points for which bit error rate is less.
- It is found by repeated experimentation which helps to demonstrate correct decoding.

# Comparison between pessimistic and optimistic noise

^
|
|
|
Y AXIS
: BER




X AXIS : Eb/N0 ---------->

Optimistic

Pessimistic

# Why probabilistic decoding?

- All the previous decoding algorithms are hard decision based decoding. If the actual code word to be decoded is not computed by the hard decision assumed, incorrect decoding will happen.
- To overcome this certainty, the probability is introduced for each bit which allows a bit to have a probability of it being 0 or 1 and thus

# Soft Decision Message Passing Algorithms

- The soft decision message passing algorithm are almost similar to the previously discussed hard decision algorithms.
- But, now the messages propagated are associated with conditional probabilities, sent as a measure of the belief for each code bit to be a 0 or a 1, given the received vector r.
- This exchange of data based on their probabilities, is termed as Belief Propagation.
- The Sum-Product Decoding Algorithm has been implemented using Belief Propagation.

# Probabilistic Decoding -Notations

- $P_r[c_i = x|r]$ is the a-posteriori probability of the communication channel where $c_i$ is a x given the value of r. $P_i = P_r[c_i = 1|r]$ and $P_r[c_i = 0|r]=1- P_i$
- $u_{ij}^{(k)} = (u_{ij}^{(k)}(0), u_{ij}^{(k)}(1))$ is the message sent by the message node $c_i$ to the check node $f_j$ in the $k^{th}$ iteration and $u_{ij}^{(k)}(x)$ is the probability (belief) value that $r_i = x$, $x \in \{0,1\}$, given the received message vector r.
- For each iteration k,
  - $u_{ij}^{(k)}(0) + u_{ij}^{(k)}(1)=1$
  - $u_{ij}^{(k)}(0)=1- P_i$ and $u_{ij}^{(k)}(1)= P_i$
- $v_{ji}^{(k)} = (v_{ji}^{(k)}(0), v_{ji}^{(k)}(1))$ is the message sent by the check node $f_i$ to the message node $c_i$ and $v_{ji}^{(k)}(x)$ is the probability (belief) value that $r_i = x$, $x \in \{0,1\}$, given the apriori probability $u_{ij}^{(k)} = (u_{ij}^{(k)}(0), u_{ij}^{(k)}(1))$ and the received message vector r.
- $v_{ji}^{(k)}(0) + v_{ji}^{(k)}(1)=1$
- $v_{ji}^{(k)}(*)$ calculates the extrinsic message relative to each and every message node, $c_i$ adjacent to the check node $f_j$.
- eg. $v_{ji}^{(k)}(0)$ calculates the probability that there are an even number of 1's on all the other message nodes, $c_m$ , $m \neq i$ adjacent to the check node $f_j$

# Probabilistic Decoding Algorithm

- If there are an even number of 1's in n message nodes, $c_0$ , $c_1$ …. $c_{n-1,}$ then
  - $P_r [c_0 \oplus c_1 \oplus .. \oplus c_n]=(1/2)+ (1/2) [\prod_{i=0}^{n-1}(1-2u_i) ]$
- The message that $c_i$ sends to $f_i$ in the kth iteration is
  - $u_{ij}^{(k)} (0)=h_{ij} (1-P_i) \prod_{j' \in Ci \neq j} v_{j'i}^{(k-1)}(0)$
  - $u_{ij}^{(k)} (1)=h_{ij} P_i \prod_{j' \in Ci \neq j} v_{j'i}^{(k-1)}(1)$
  - where $C_i$ = {j:$c_i$ is adjacent to $f_j$ in the Tanner graph}. $h_{ij}$ is a constant selected in a manner such that $u_{ij}^{(k)} (0) + u_{ij}^{(k)} (1)=1$
- At each iteration at each message node $c_i$
  - $Q_i^{(k)} (0)=h_i (1-P_i) \prod_{j \in Ci} v_{ji}^{(k-1)}(0)$
  - $Q_i^{(k)} (1)=h_i P_i \prod_{j \in Ci} v_{ji}^{(k-1)}(1)$
- $Q_i^{(k)} (x)$ is the effective probability that $c_i = x$, $x \in \{0,1\}$ at each iteration k.
- Estimation in the kth iteration is $c_i = 1$ if $Q_i^{(k)} (1) >= Q_i^{(k)} (0)$, else $c_i = 0$.
- Syndrome calculation is same as in hard decision decoding.
- If syndrome is 0, algorithm terminates , else the algorithm runs for a specified number of iterations.

# Error Correction for probabilistic decoding

- Number of code words for each noise point is 10. All 0 code words are encoded.
- The noise points are
  - EbN0 = [0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6]
- 5 iterations are carried during decoding.
- Eg. For Noise point 5 i.e 2.5 dB
- Number of non zeros after decoding 47.
- The bit error rate should be 47/2000.
- The bit error rate is 2.350000e-02.

# Standard Deviation and Mean for probabilistic decoding

| Noise (in dB) | 0.5 | 1 | 1.5 | 2 | 2.5 | 3 |
|---|---|---|---|---|---|---|
| Standard Deviation | 0 | 0.5964092 | 0.2542696 | 0.07692198 | 0.0358956 | 0.0457442 |
| Mean | 1.130500 | 0.534775 0 | 0.251066 7 | 0.124162 5 | 0.075000 00 | 0.050141 67 |

Implication that much of the data over individual noise is spread around the mean.

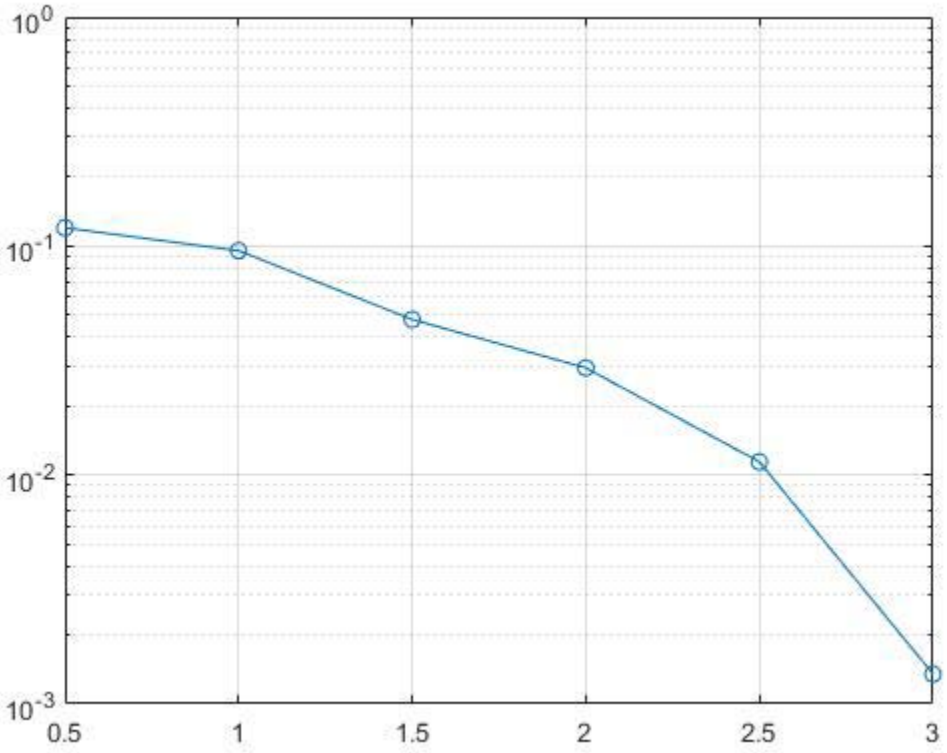
- The overall standard deviation : 0.04068366
- The overall mean : 0.02468333
- 12 Test cases are considered and then averaged over.

# Standard Deviation and Mean for probabilistic decoding

| Noise (in dB) | 3.5 | 4 | 4.5 | 5 | 5.5 | 6 |
|---|---|---|---|---|---|---|
| Standard Deviation | 0.0461227 4 | 0.0455931 7 | 0.0443983 3 | 0.0431335 4 | 0.0418833 0 | 0.0406836 6 |
| Mean | 0.0427000 0 | 0.0370250 0 | 0.0329111 1 | 0.0296200 0 | 0.0269272 7 | 0.0246833 3 |

# Bit error rate to noise for probabilistic decoding

^
|
|
|
Y AXIS
: BER




X AXIS : Eb/N0 ----------->

# Why log likelihood decoding?

- The previous algorithm required a lot of multiplications to be done. But multiplications are costly operations to implement.
- Thus, the Log-likelihood domain is implemented to turn the multiplications into additions which are more cheaper to implement in both software and hardware.

# Log Likelihood decoding algorithm

- The Likelihood ratio provides the belief for each code bit ci to be a 0 or a 1.
  - $L_i = (P_r[c_i = 0|r] / P_r[c_i = 1|r]) = (1- P_i)/ P_i$
- Taking logarithm on both sides, the log-likelihood ratio is obtained.
  - $l_i = \ln(L_i) = \ln((1- P_i)/ P_i)$
- $l_i$ is used in the modified Sum-Product decoding algorithm turns multiplications into additions.
- The message that $c_i$ sends to $f_i$ in the kth iteration is
  - $m_{ij}^{(k)} = \ln[u_{ij}^{(k)}(0)/ u_{ij}^{(k)}(1)]$
    $= \ln[((1- P_i)/ P_i) \prod_{j' \in Ci \neq j} [v_{j'i}^{(k-1)}(0)/ v_{j'i}^{(k-1)}(1)]]$
    $= l_i + \sum_{j' \in Ci \neq j} m_{j'i}^{(k-1)}$
  - $e^{mij} = (1- u_{ij}(1))/u_{ij}(1)$, or $u_{ij}(1)=(1/(1+ e^{mij}))$
  - Thus $(1-2u_{ij}(1))= (e^{mij} -1)/(e^{mij} +1)=\tanh(m_{ij}/2)$
- The message that $c_i$ sends to $f_i$ in the kth iteration is
  - $m_{ij}^{(k)} = \ln[v_{ij}^{(k)}(0)/ v_{ij}^{(k)}(1)] = \ln[(1/2)+ (1/2)[\prod_{i' \in Fj \neq i}(1-2u_{i'j}^{(k-1)}(1)]/(1/2)-(1/2)[\prod_{i' \in Fj \neq i}(1-2u_{i'j}^{(k-1)}(1))] = \ln[1+ [\prod_{i' \in Fj \neq i}(\tanh(m_{i'j}^{(k-1)}/2)]/1- [\prod_{i' \in Fj \neq i}(\tanh(m_{i'j}^{(k-1)}/2)]]$

# Log Likelihood decoding algorithm

- At each iteration and at each message node $c_i$, $l_i^{(k)}$ is computed instead of computing $Q_i^{(k)}(x)$, $x \in \{0,1\}$ . If $l_i^{(k)} >= 0$, then $c_i = 0$, otherwise $c_i = 1$.
- $l_i^{(k)} = \ln [Q_i^{(k)}(0)/ Q_i^{(k)}(1)] = l_i^{(0)} + \ln[\prod (v_{ji}^{(k)}(0)/ v_{ji}^{(k)}(1)] = l_i^{(0)} + \sum_{j \in Ci} \ln[(v_{ji}^{(k)}(0)/ v_{ji}^{(k)}(1)] = l_i^{(0)} + \sum_{j \in Ci} m_{ji}^{(k)}$
- In reality, the log likelihood algorithm or one of its variants is executed for a maximum number of iterations or until the passed likelihoods are closed to certainty, whichever comes first. A certain likelihood is $l_i = \pm\infty$ where $P_i \rightarrow 0$ when $l_i \rightarrow \infty$ and $P_i \rightarrow 1$ when $l_i \rightarrow -\infty$.

# Time complexity of log-likelihood algorithm

- O(N) time complexity, where N = block length of the code
- In each of the iterations, messages traverse between check nodes and message nodes through the edges and since an edge is equivalent to a 1 of an entry in the sparse parity check matrix, therefore the number of traversals through the edges are small.
- Moreover, for a fixed number of iterations, each edge is traversed a maximum number of times which is a constant. If the number of nodes grow increase linearly with the block length of the code, then the total number of operations also grow linearly with the block length of the code.
- The Sum-Product decoder is entirely independent of the channel used, though the messages passed, while the algorithm runs, are completely dependent on the channel

# Error Correction for log likelihood decoding

- Number of code words for each noise point is 10. All 0 code words are encoded.
- The noise points are
  - EbN0 = [0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6]
- 5 iterations are carried during decoding.
- Eg. For Noise point 5 i.e 2.5 dB
- Number of non zeros after decoding 47.
- The bit error rate should be 47/2000.
- The bit error rate is 2.350000e-02.

# Standard Deviation and Mean for log likelihood decoding

| Noise (in dB) | 0.5 | 1 | 1.5 | 2 | 2.5 | 3 |
|---|---|---|---|---|---|---|
| Standard Deviation | 0 | 0.5896917 | 0.2138452 | 0.1071708 | 0.0406093 | 0.0451034 |
| Mean | 1.200500 | 0.5370250 | 0.2309833 | 0.1391750 | 0.0813100 | 0.0529083 |

Implication that much of the data over individual noise is spread around the mean.

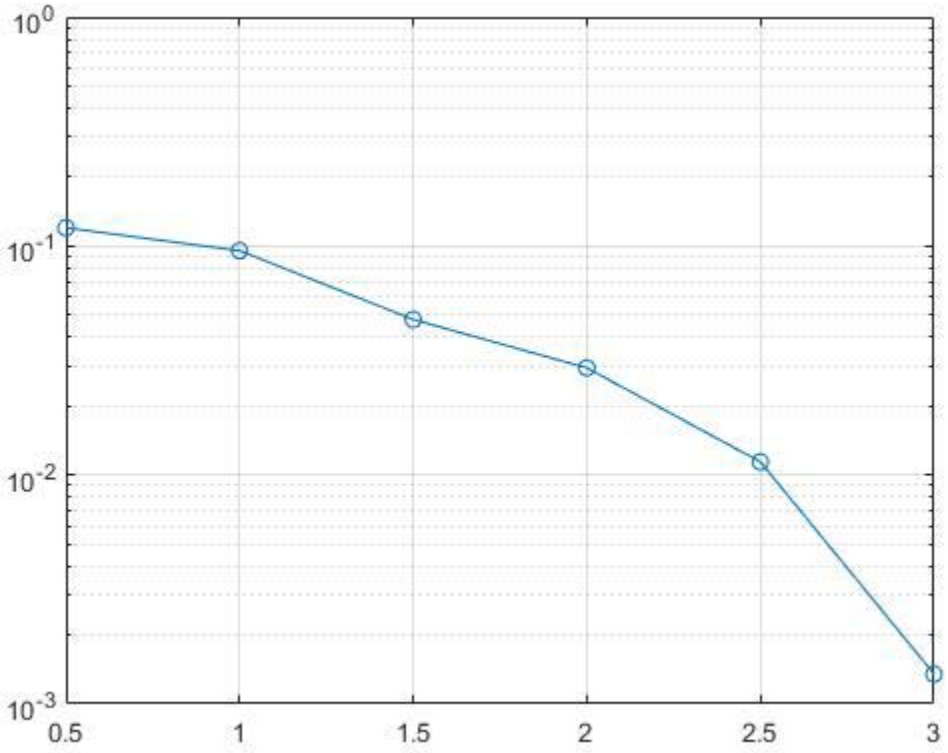
- The overall standard deviation : 0.04158134
- The overall mean : 0.02544167
- 12 Test cases are considered and then averaged over.

# Standard Deviation and Mean for log likelihood decoding

| Noise (in dB) | 3.5 | 4 | 4.5 | 5 | 5.5 | 6 |
|---|---|---|---|---|---|---|
| Standard Deviation | 0.0473831 7 | 0.0464994 9 | 0.0453182 8 | 0.0440525 2 | 0.0427936 5 | 0.0415813 4 |
| Mean | 0.0436142 9 | 0.0381625 0 | 0.0339222 2 | 0.0305300 0 | 0.0277545 5 | 0.0254416 7 |

# Bit error rate to noise for log likelihood decoding

^
|
|
|
Y AXIS
: BER




X AXIS : Eb/N0 ----------->

# Why Not minimum distance based decoding?

- NETWORK MATTERS
- Low SNR (Signal to Noise Ratio) in case of minimum distance based decoding, a problem in deep space communication.
- Neighbours came to the rescue which incorporates the minimum distance concept too, showing a better SNR.
- Instead of an overall minimum distance, it is important to see how to achieve maximum likelihood decoding based on only those nodes that influence . This is what such a decoder tries to achieve.

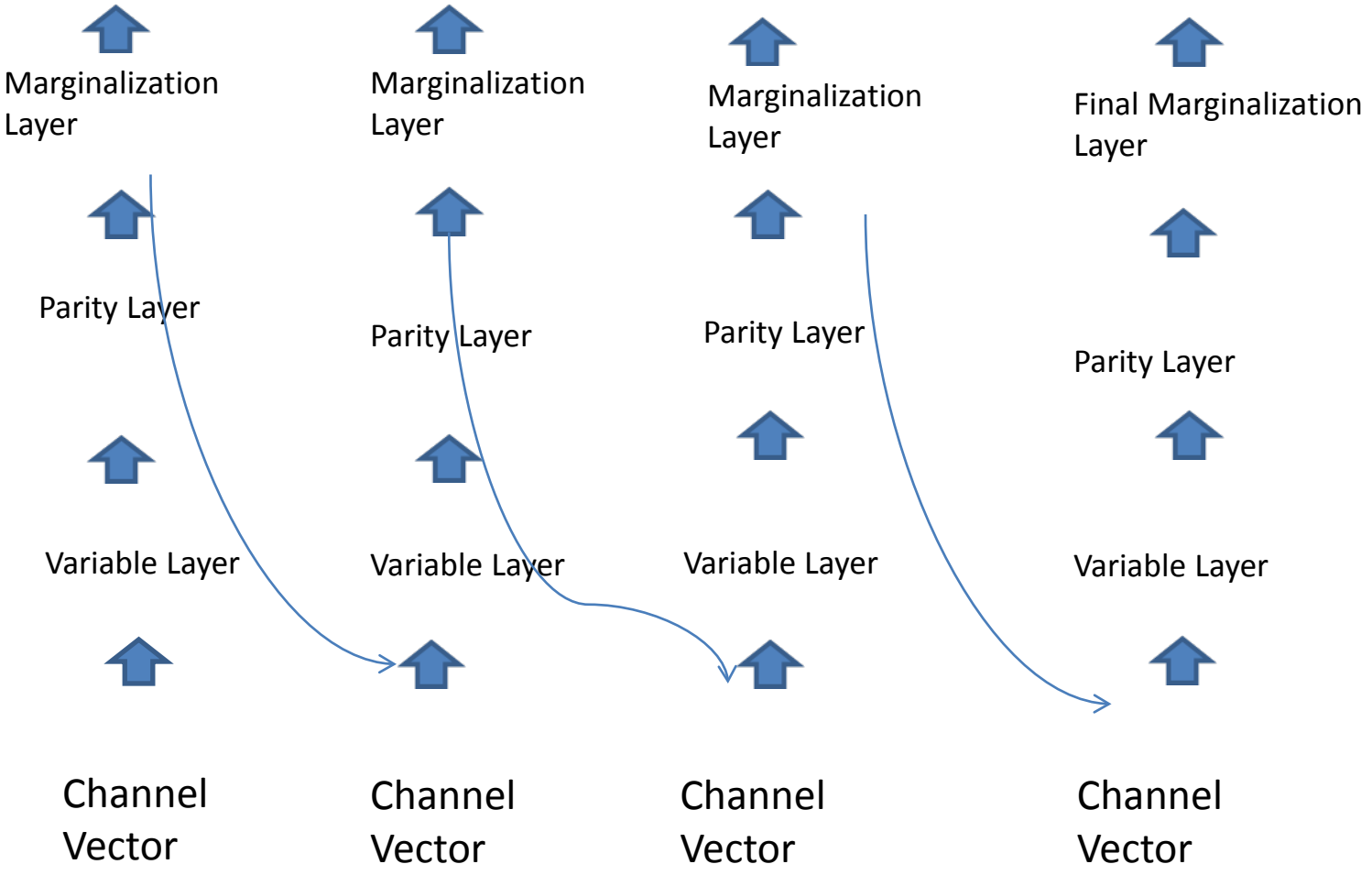
# Why Deep Learning to LDPC Decoding?

- an improved neural network architecture [6] that achieves similar results to [5] with less parameters and reduced complexity.
- The main difference compared to [5] is that the offset min-sum algorithm is used instead of the sum-product algorithm, thus eliminating the need to use multiplications.

# Deep Learning to LDPC Decoding

- that deep learning methods can improve the BP decoding of HDPC codes using a weighted BP decoder. [5]
- The BP algorithm is formulated as a neural network and it is shown that it can improve the decoding by 0.9dB in the high SNR regime.
- it is sufficient to train the neural network decoder using a single code word (e.g., the all-zero code word), since the architecture guarantees the same error rate for any chosen transmitted code word.

# Recurrent Neural Network Architecture with unfold 4 which corresponds to 4 full BP iterations



# SMART DECODING

# Neural Network Algorithm

- Two approaches have been adopted
  - 1)Sequential training
  - 2)Aggregate training
- Feed-forward neural network is used capitalising on the architecture of low density parity check codes. The decoding problem is considered to be a regression problem.
- The decoding in training is done using any of the previous decoding algorithms. (Log-likelihood has been considered for experimentation).
- 1) Sequential training:- For each received codeword for a particular frame and a particular noise point, training is done with the received codeword as input and decoded codeword as output which tunes the network continuously thus correcting all errors.
- 2) Aggregate training:- For all the received code words and the decoded code words which satisfy the parity check equations, training is done.

# Sequential training parameters

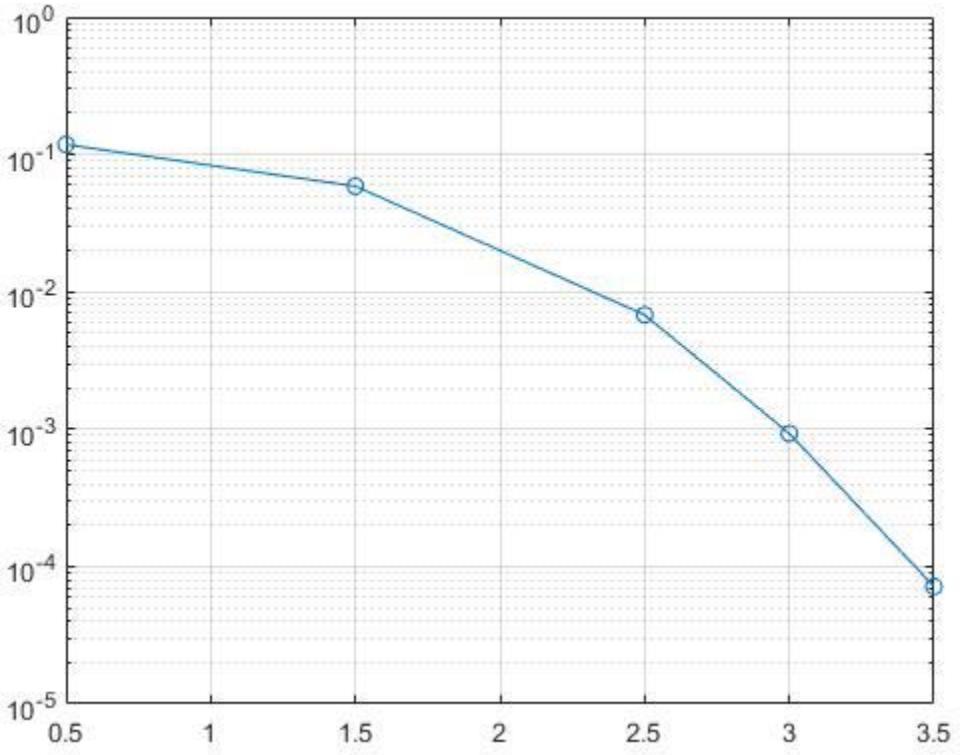
- Number of hidden layers:-10
- Number of epochs:- 100
- Batch size:-7
- Number of validations:-4
- Initial Learning rate =0.01
- Factor for dropping learning rate=0.1
- Number of noise points in training:  10
  - EbN0 = [0.5 1.5 2.5 3 3.5 4.5 5 5.5 6 7]
- Number of training frames:7
- Number of noise points in training:  5
  - EbN0 = [0.7 1 2 4 5.2];
- Number of testing frames:3
- Network performance function (msereg):- It measures network performance as the weight sum of two factors: the mean squared error and the mean squared weight and bias values.

# Sequential training mechanism

- The neural network is configured for each input received message and each output decoded message, one at a time.
- Data Division is done randomly.
- Training is done using the Levenberg-Marquardt method in matlab.
  - It minimizes a function which is a sum of squares in least square problem
- The received codeword is given as input while the decoded codewords from log-likelihood domain are given as outputs.
- Tuning is done of the input parameter which improves regression.
- Since noise is incorporated in received codeword, the received codewords can be different for the same encoded codeword.
- Unknown received codewords are given as inputs during testing to obtain the decoded codewords.
- Note:- All the decoded codewords of the parity check matrix are considered.

# Bit error rate to noise for Sequential Training and tuning

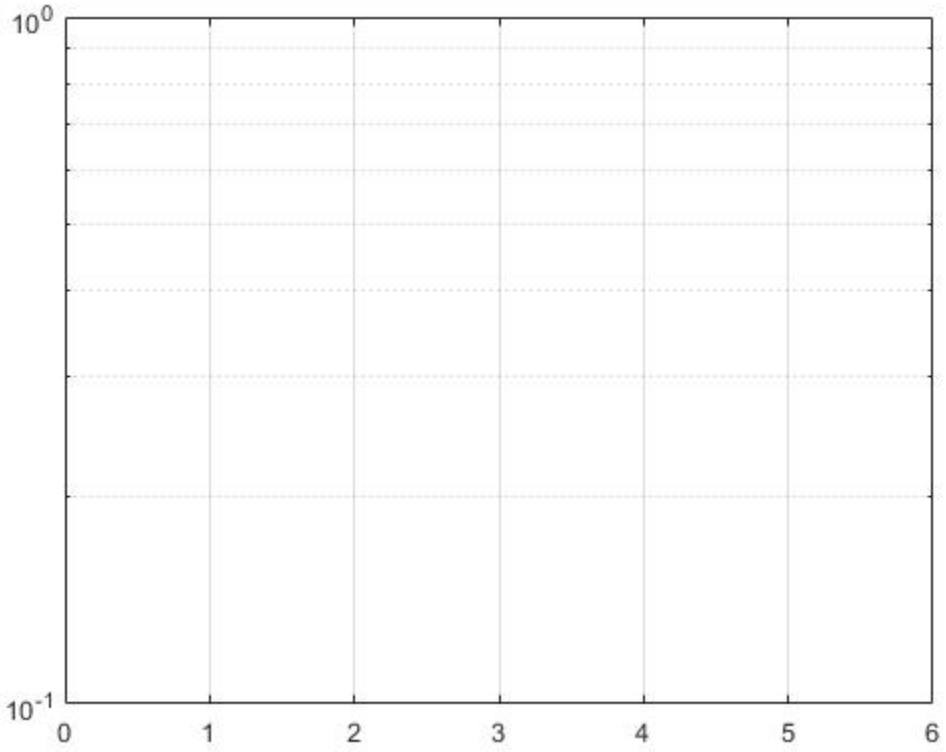
^
|
|
|
Y AXIS
: BER




X AXIS : Eb/No ----------->

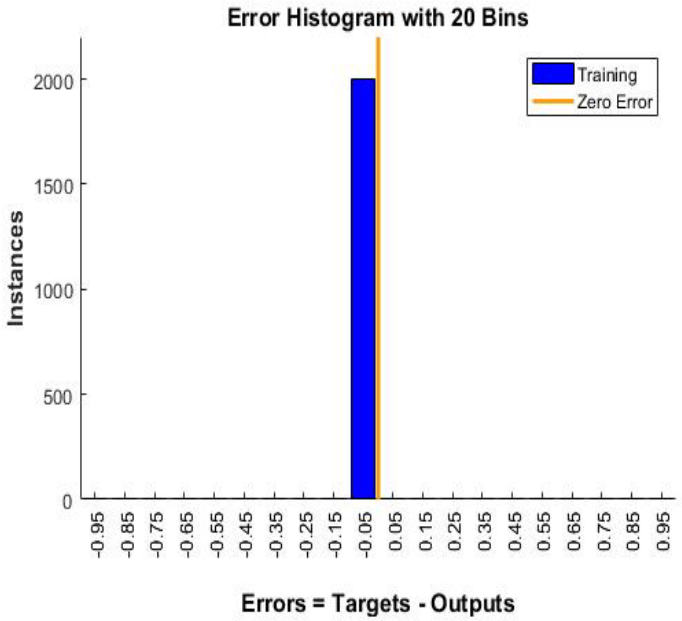
# Bit error rate to noise for Testing the above algorithm

^
|
|
|
Y AXIS
: BER




X AXIS : Eb/No ---------->

# Neural Network and error histogram

# Aggregate training parameters

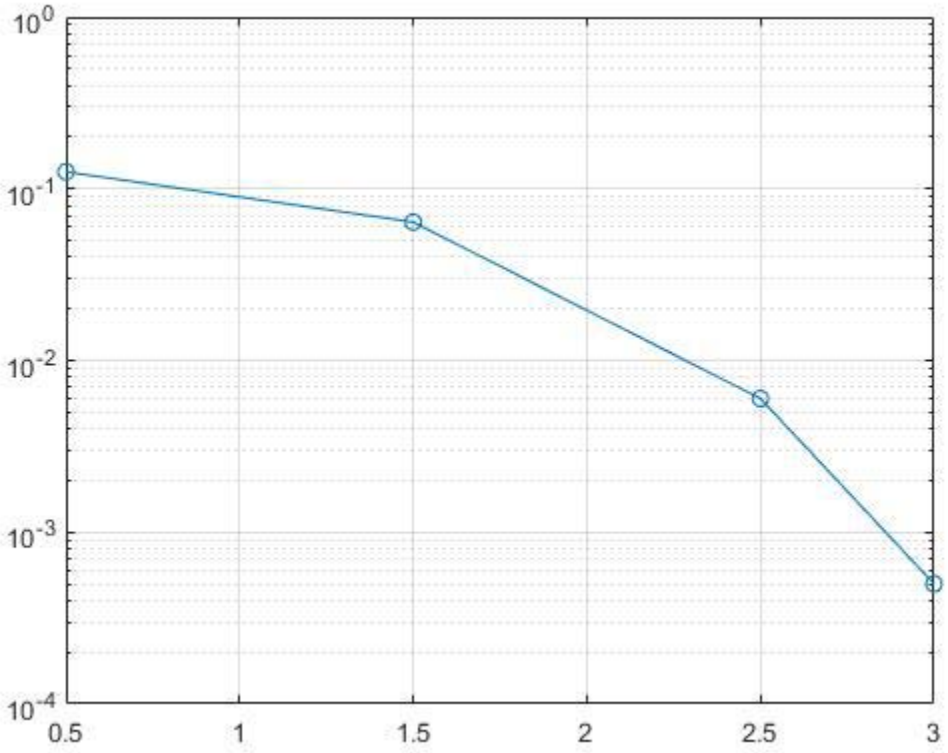
- Number of hidden layers:-10
- Number of inputs:-25
- Number of layers:-25
- Number of epochs:- 4
- Batch size:-25
- Number of validations:-4
- Initial Learning rate =0.01
- Factor for dropping learning rate=0.1
- Number of noise points in training:  5
  - EbN0 = [0.5 1.5 2.5 3 3.5];
- Number of training frames:5
- Number of noise points in training:  5
  - EbN0 = [0.3 0.7 1 1.4 1.8];
- Number of testing frames:5
- Network performance function (msereg):- It measures network performance as the weight sum of two factors: the mean squared error and the mean squared weight and bias values.

# Aggregate training mechanism

- The neural network is configured for all the input received messages and output decoded messages.
- Data Division is done randomly.
- Training is done using the Levenberg-Marquardt method in matlab.
  - It minimizes a function which is a sum of squares in least square problem
- The 25 received codewords are given as input while all the decoded codewords from log-likelihood domain are given as outputs.
- Since noise is incorporated in received codeword, the received codewords can be different for the same encoded codeword.
- Unknown received codewords are given as inputs during testing to obtain the decoded codewords.

# Bit error rate to noise for Aggregate Training and tuning

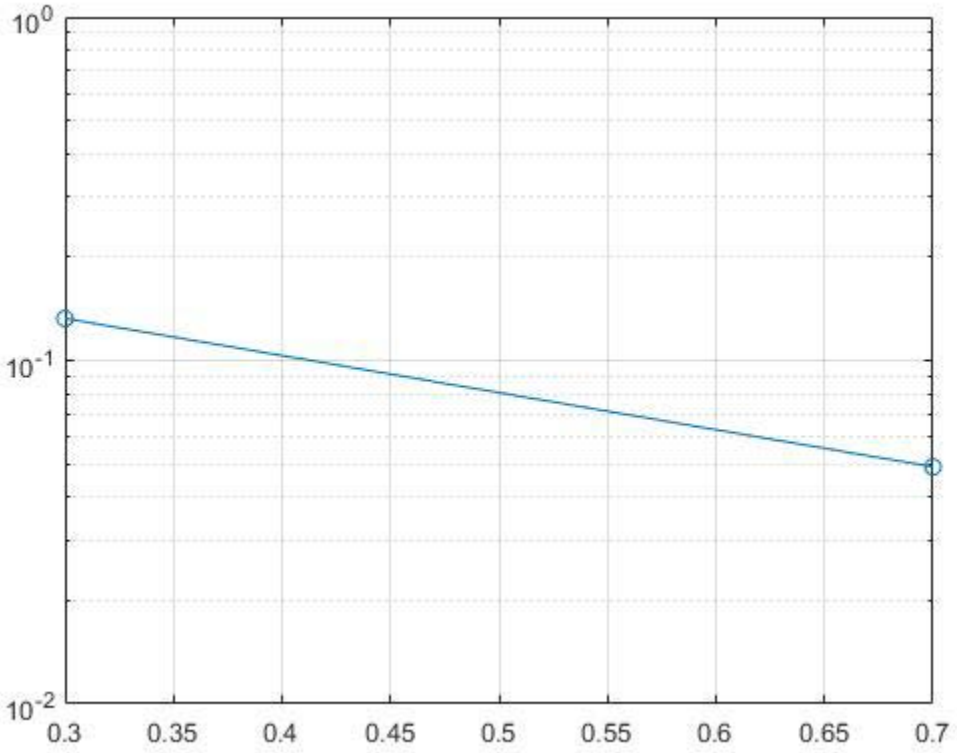
^
|
|
|
Y AXIS
: BER




X AXIS : Eb/No ---------->

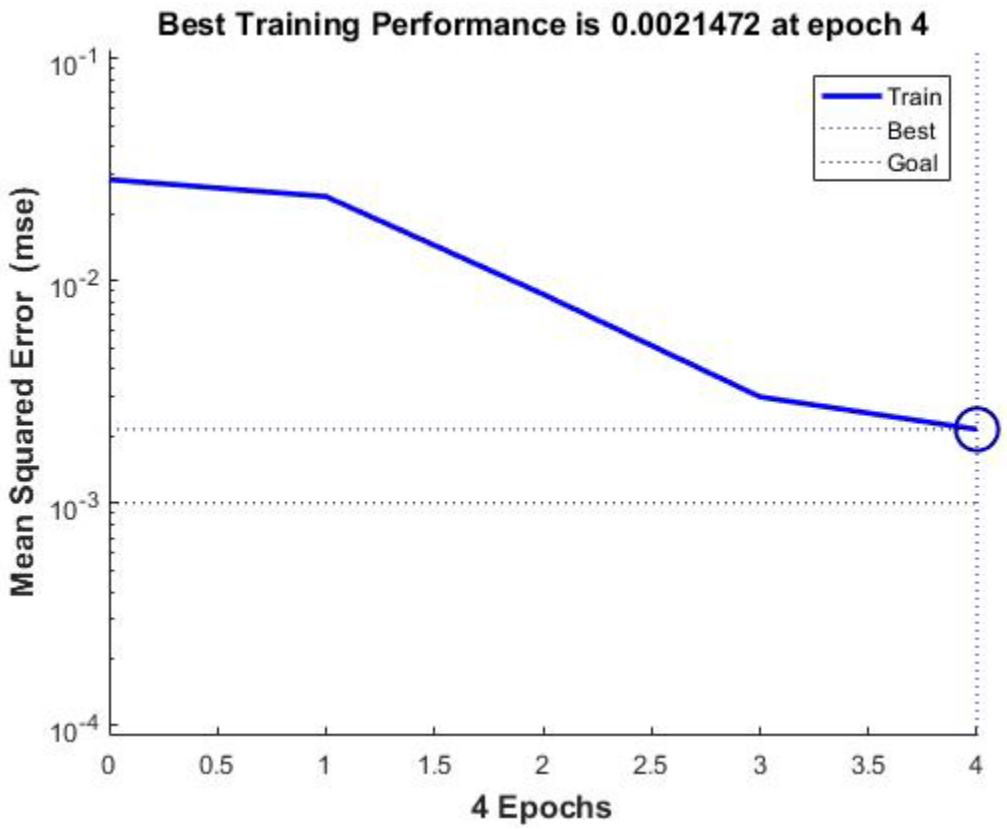
# Bit error rate to noise for Testing the above algorithm

^
|
|
|
Y AXIS
: BER

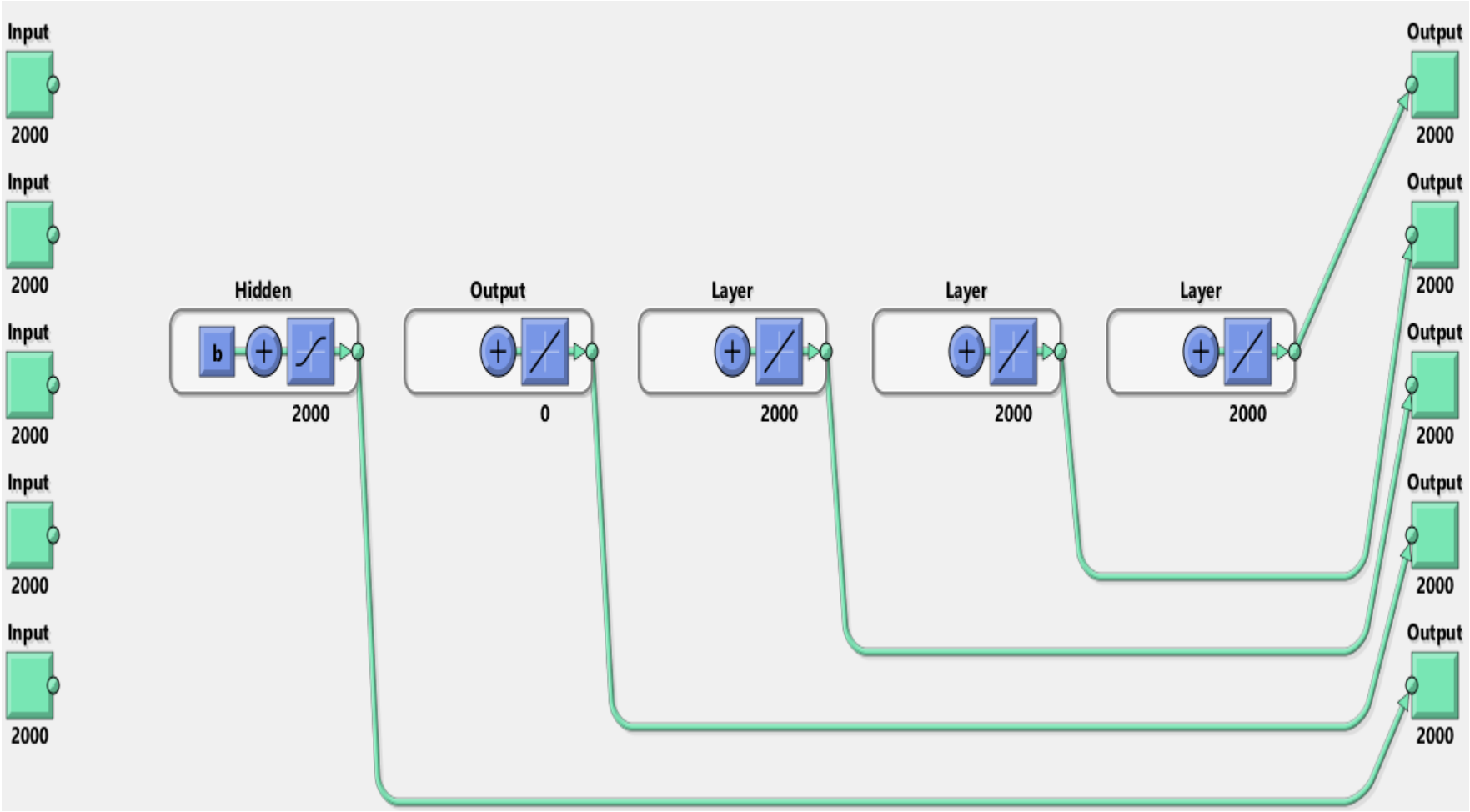



X AXIS : Eb/No ----------->

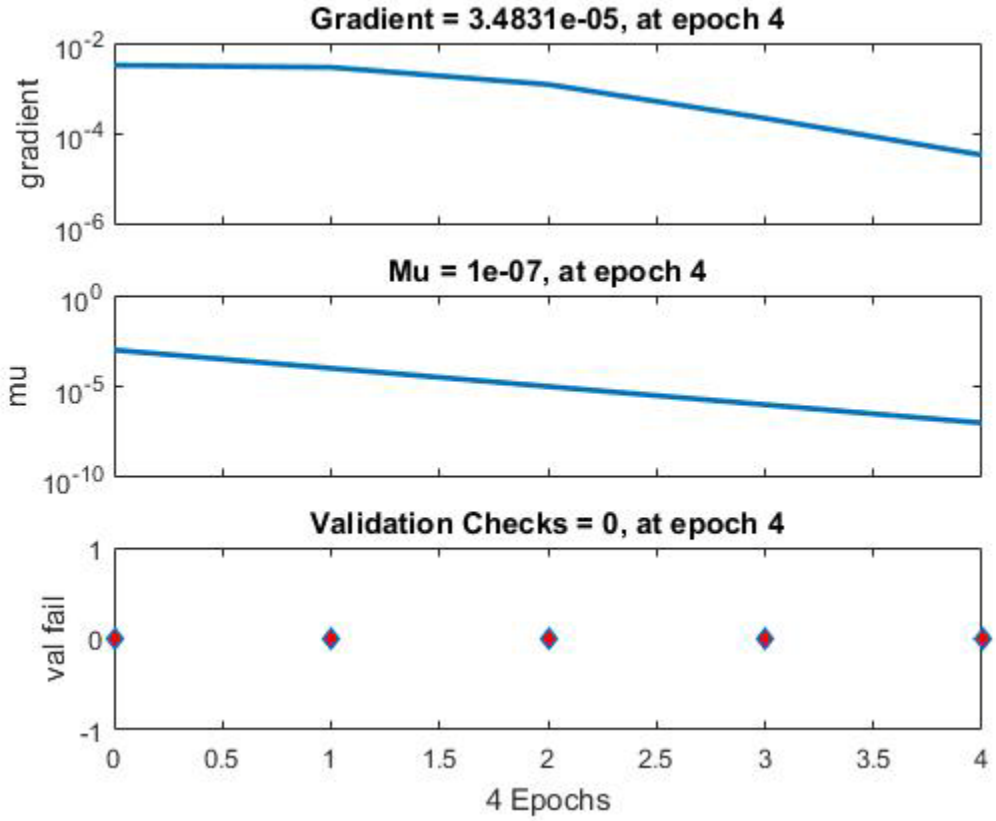
# Neural Network Training Performance



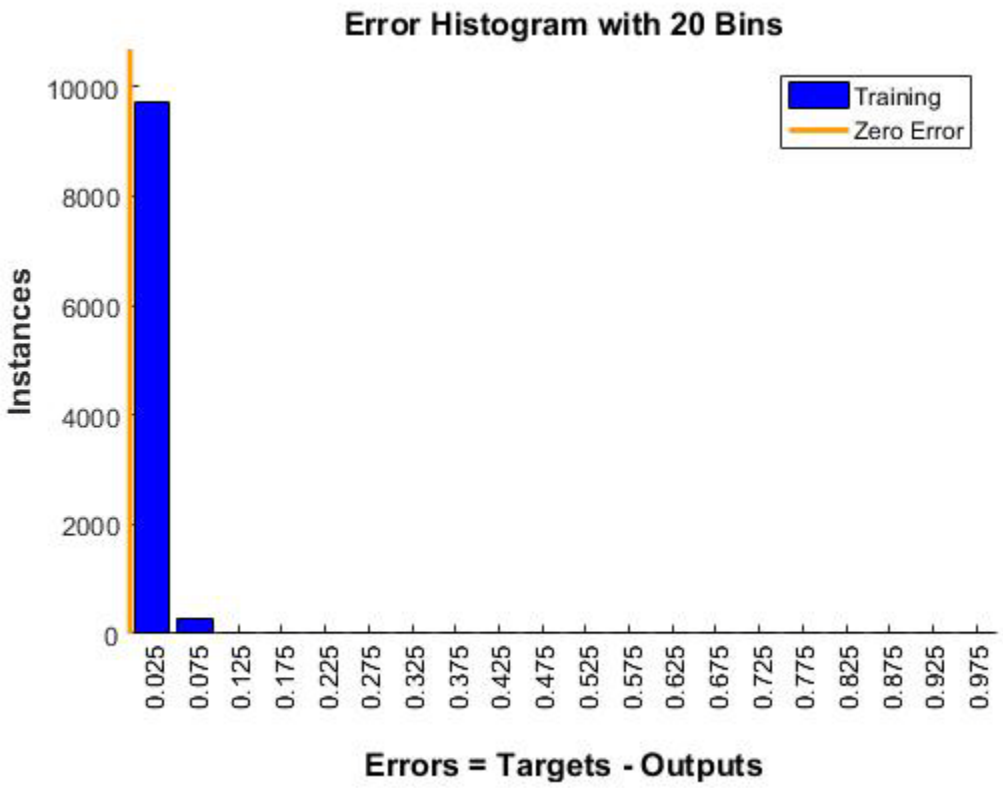
# Neural network structure for 5 inputs



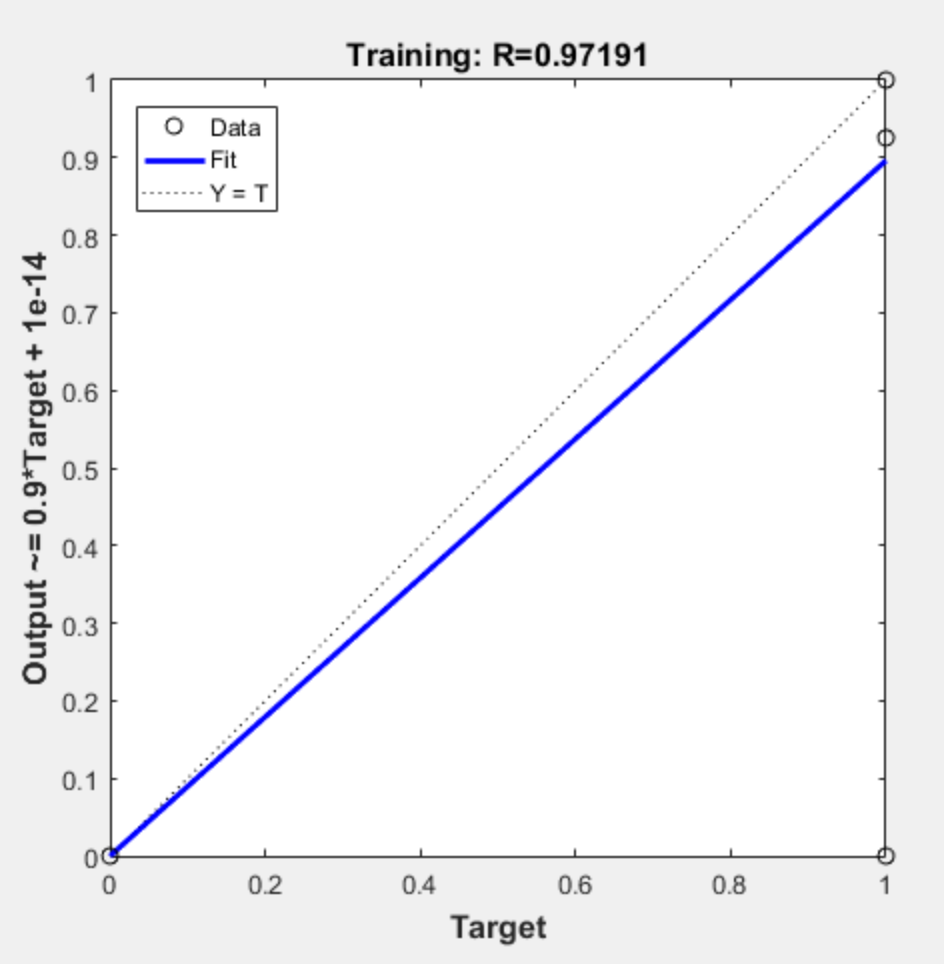
# Neural network training state



# Error histogram



# Regression


Training: R=0.97191
Data
Fit
Y = T
Output ~= 0.9*Target + 1e-14
Target
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.2
0.4
0.6
0.8
1

# Error Correction for aggregate training method during testing

- Number of code words for each noise point is 5. All 0 code words are encoded.
- 5 iterations are carried during decoding.
- Eg. For Noise point 2 i.e 0.7 dB
- Number of non zeros after decoding 98
- The bit error rate is 0.04.900000
- The standard deviation and mean of the bit error rates for Noise 2 are : 0.05781176 and 0.036300.

# Standard Deviation and Mean for aggregate training method during testing

| Noise (in dB) | 0.3 | 0.7 | 1 | 1.4 | 1.8 |
|---|---|---|---|---|---|
| Standard Deviation | 0.05629121 | 0.05781176 | 0.05781176 | 0.05781176 | 0.05781176 |
| Mean | 0.03830000 | 0.03630000 | 0.03630000 | 0.03630000 | 0.03630000 |

Implication that much of the data over individual noise is spread uniformly.

- The overall standard deviation : 0.05781176
- The overall mean : 0.03630000
- 5 Test cases are considered and then averaged over.

# Limitation of neural network approach

- Relevant neural network decoding has been done in high density parity check codes but **NOT** in low density parity check codes. So no literature is available for comparison.
- In existing works, the learning lacks diversification.
- Convolutional neural network approaches have not been attempted which resemble more to the neighbourhood consultation in belief propagation.

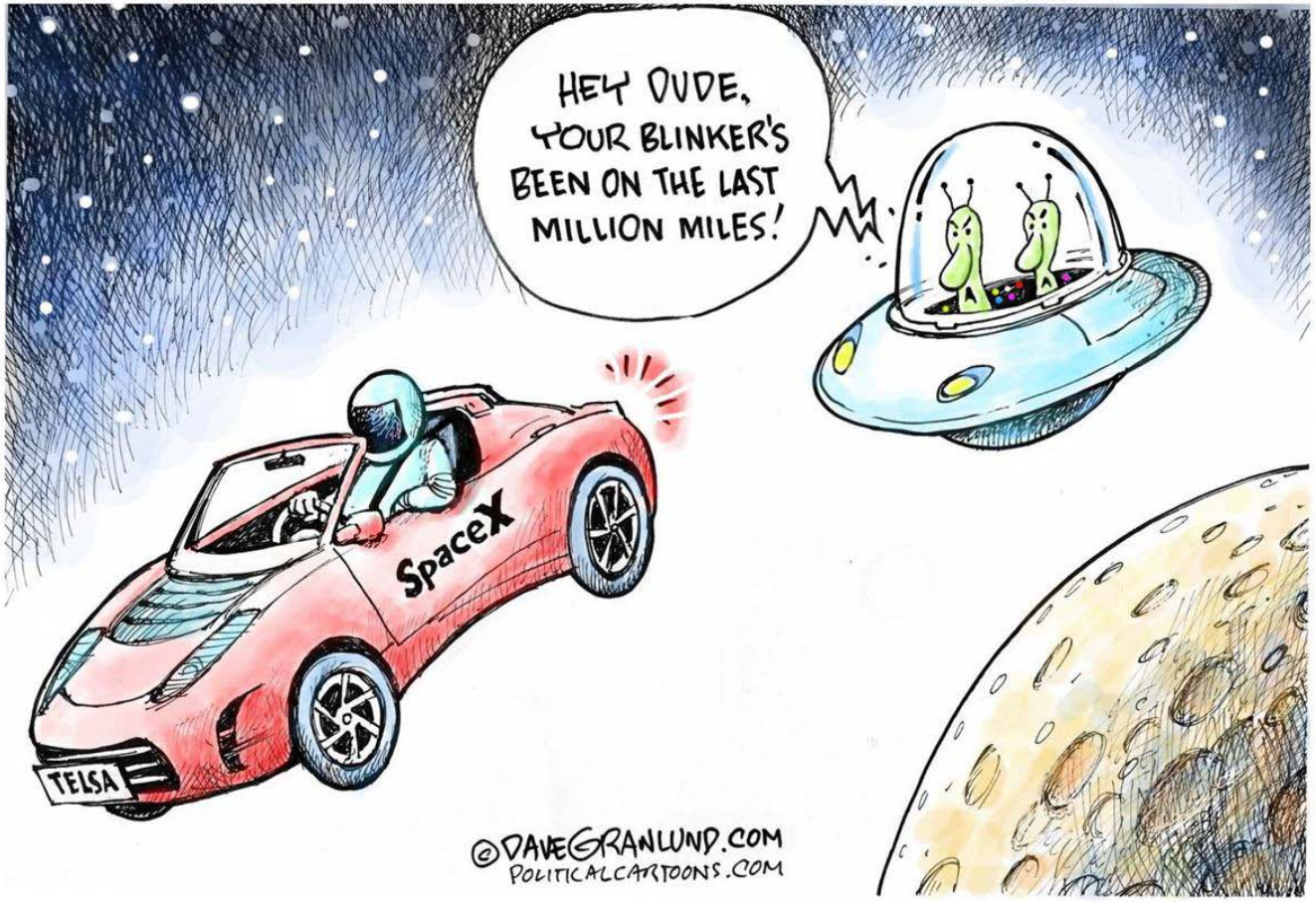
# Future Work

- Training can be done with multiple valid code words during encoding as inputs along with their computed Hamming distances to diversify the learning.
- This diversification can also be done by training with multiple noise points also in addition to multiple code words as inputs.
- Resource dependent work
  - GPU accessibility needed for training with multiple layers and multiple outputs
  - This can also be shortchanged by assigned weights to the Tanner graph which will however be inefficient compared to the above.
- Convolutional neural network architecture should be implemented.
- Extension to irregular LDPC codes
- Increase the number of frames and the number for further experimentation.

# WHAT IF?

- All our work has been based on decoding and training the received messages.
- What if, the encoder is itself unknown?
- Encoders can be trained and thus unknown encoders can be predicted!!!
- We don’t know the encoders of aliens!!
- So…….

# Maybe true, after training encoders!!



# References


- [1] R.G. Gallager, “Low Density Parity Check Codes,” MIT Press,Cambridge,1963.
- [2] B.M.J. Leiner, “LDPC Codes – a Brief Tutorial”, April,2005
- [3] https://sites.google.com/site/bsnugroho/ldpc
- [4] E. Nachmani, Y. Be’ery, and D. Burshtein, “Learning to decode linear codes using deep learning,” in 54’th Annual Allerton Conf. On Communication, Control and Computing, September 2016, arXiv preprint arXiv:1607.04793.
- [5] L. Lugosch and W. J. Gross, “Neural offset min-sum decoding,” in 2017 IEEE International Symposium on Information Theory, June 2017, arXiv preprint arXiv:1701.05931.
- [6] Mikale Simberg, “Linear-time encoding and decoding of low-density parity-check codes”, Aalto University, Master’s Thesis, 2015