On Machine Learning Enhancements to LDPC Decoding

Satya Kumar Vankayala[1], Tirtankar Mittra[1], Seungil Yoon[2], Thirumulanathan D[3]
[1]Networks S/W R&D Group, Samsung R&D Institute, Bengaluru, India
[2]Network Business, Samsung Electronics, Suwon, South Korea
[3]IIT Kanpur, India

***Abstract*—Low Density Parity Check (LDPC) codes, invented by Gallager, have revolutionized communications by being capacity approaching codes while possessing a relatively low decoding complexity. For these reasons, they were deployed in 4G communication systems as the error correcting code. With the advent of 5G communications, vRAN and O-RAN technologies, the usual LDPC decoder does not perform well with respect to low latency requirements. Part of the reason is attributed to the soft decoder that involves the computation of non-linear functions. In this paper, we discuss a piecewise linear (PL) as well as a machine learning (ML) approximation of the LDPC soft decoder. We show that our designs bring down the decoder complexity with negligible effects on performance and are suitable for implementation in vRAN/O-RAN.**

I. Introduction

Fifth generation (5G) communication technology [1] aims to settle the problem of many users seeking high data rates. With the advances in Internet of Things and massive increase in number of connected users, the existing 4G communication was found lacking in providing a reliable service. Some of the ways 5G attempts to solve this problem include but not limited by

1) Exploiting higher frequency bands, namely millimeter waves (mmWave) [2] in order to have more system resources. In fact the available bandwidth of 5G downlink is ten times that of 4G.
2) Utilization of virtual radio access network (vRAN), cloud radio access networks (cRAN) and operator defined RAN (O-RAN) [3] architectures which offload computationally intensive operations to the cloud or a centralized server. This architecture speeds up several existing communication modules, in some cases replacing them wholly in software.
3) Using capacity achieving error correcting codes to improve communications and data rate.

In this paper, we focus on the third aspect.

Given a communication channel, the channel capacity is the maximum data rate for which there exists a transmission scheme (codeword lengths may be very large if required), that ensures errors can be brought under control. An error correcting code involves padding the message bits to be transmitted with carefully designed parity bits which allow for certain errors to be detected as well as corrected. Since we are transmitting redundant bits, the overall data rate is reduced. However there exist a very special class of error correcting codes that have the capacity approaching property, i.e. they add redundancy in such a way that data rates close to capacity may be achieved. In 5G communications, two such codes have been chosen for incorporation namely low density parity-check (LDPC) codes [4] (for the data channels) and polar codes [5] (for the control channels).

LDPC codes, developed by Gallager [4] in early 1960's were one of the first capacity approaching codes ever designed. However at the time, they were considered impractical and technology had not progressed enough to utilize them. Moreover efficient generation of parity check matrices as well as decoding methods for LDPC were not known until the work of Tanner [6] as well as Richardson et. al [7].

A. Main Contributions

The commonly used method of LDPC decoding involves belief propagation which is an iterative process. At each stage, a likelihood function is updated that involves very complicated functions. We propose two ways to circumvent this issue.

1) Piecewise linear approximation of likelihood function update formula [8].
2) Machine learning based approximation of a popularly used min-max approximation [9].

Some of the salient features of these approximations are

1) **Controllable accuracy**. Depending on system requirements, we can have a more accurate representation of the function as close as required.
2) **Simple Hardware**. The piecewise linear approximation only requires simple adders, multipliers and comparators. Once trained to desired accuracy, it may be deployed as is.
3) **Better space and time complexity**. The original function required a large look up table or saving of several polynomial coefficients via Taylor approximation to work. The former has poor space complexity and the latter time. The proposed methods mitigate these issues.

Further speedups are possible if these approximations are implemented in vRAN / O-RAN architecture. Moreover, the O-RAN module features a RAN intelligent controller (RIC) module where these methods may be implemented.

The paper is organized as follows. In Section II, we give a quick overview of LDPC codes and develop notation. We also discuss the functions that are being approximated by our methods. Next, we provide details on the three regression methods

in Section III. The results hence obtained are discussed in Section IV, along with comparisons. Finally we conclude the paper.

II. Preliminaries of LDPC codes

A (n, k) linear block code [10] consists of a generator matrix $\mathbf{G} \in \{0,1\}^{k\times n}$ which takes a k-bit message vector $\mathbf{m}$ as input and outputs an n-bit codeword $\mathbf{c} = \mathbf{mG}$. The generator matrix is designed so as to impart certain minimum Hamming distance between codewords and consequently some error correcting properties to the generated codeword. Using $\mathbf{G}$, a matrix $\mathbf{H} \in \{0,1\}^{q\times n}$, where $q \geq n-k$, called the parity-check matrix is constructed whose nullspace is the set of codewords. This means for any codeword $\mathbf{c}$, we have $\mathbf{cH}^T = 0$. When a non-codeword is given instead, a non-zero syndrome vector is generated that will enable us to detect and correct errors (see [10] for more details). When the parity check matrices are large, the decoding complexity also increases considerably.

LDPC codes are a special class of linear block codes where the parity-check matrices are large but also sparse i.e. more than half the entries are 0. This allows for efficient representation as well as decoding algorithms e.g. message passing over Tanner graphs [6], to be used that take advantage of the sparse nature of $\mathbf{H}$. A regular LDPC code is characterized by a parity check matrix $\mathbf{H}$ that satisfies

1) Number of ones in each row is n_r; in each column is n_c.
2) Number of ones in common between any two columns are at most one.
3) Both n_r and n_c are small compared to the dimensions of $\mathbf{H}$. In particular, we require $n_r << n$ and $n_c << q$.

An example of an $n_r = 4$, $n_c = 2$ regular LDPC code is

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

where we see that n_r and n_c satisfy the regularity requirements.

We only consider regular LDPC codes in our purview. While there are several methods for generating good LDPC matrices, it doesn't matter which method is used in our analysis as the focus will be on soft decoder design. Next we discuss the LDPC encoder and decoder.

To make decoding easier, the generator matrix is often described in a systematic form wherein $\mathbf{G}$ is of the form $[\mathbf{I}\ \mathbf{P}]$, where P is called the parity generator matrix. As a result, the first k bits of the codeword are the same as the message bits and the remaining $n-k$ bits are referred to as the parity bit which aid in error detection and correction.When designed in systematic form, there is no difficulty in extracting the message part. The main complexity lies in the error detection and correction part (for instance in the presence of noise where bits get flipped), especially if the size of parity check matrices are large.

A. LDPC Encoder

Once a suitable parity check matrix $\mathbf{H}$ is chosen, there are several ways to encode the data [7], [11]. One method discussed in [7] involves the following steps.

1) Perform Gaussian elimination on $\mathbf{H}$ to bring it to an upper triangular matrix. This may involve column swapping.
2) The last k bits of the codeword $\mathbf{c}$ are set to message bits. That is, for $1 \leq i \leq k$, $c_{n-k+i} = m_i$.
3) Perform backward recursion starting from $p = n-k$ to get

$$c_p = -\sum_{j=p+1}^{n-k} H_{p,j}c_j - \sum_{j=p+1}^{k} H_{p,j-n+k}m_j \tag{2}$$

where $H_{p,j}$ is the element on the p^{th} row and k^{th} column of $\mathbf{H}$.

B. LDPC Decoder

Efficient LDPC decoding is done using either message passing (MP) [12], belief propagation (BP) methods [8] on Tanner graphs or extrinsic information transfer (EXIT) charts [13]. In this paper, we will focus on BP methods. We will also recall the basic system for the sake of clarity.

Assume that the codeword generated from the previous section is sent through a binary input memoryless channel (BIMC) with channel pmf $W(y|x)$, where x is the channel input and y is the corresponding output. The input is assumed to satisfy an average power constraint of P i.e. $\mathbb{E}[X^2] \leq P$. Given a codeword $\mathbf{c} \in \mathcal{C}$ to be sent on the BIMC, we first perform binary phase shift keying (BPSK) on $\mathbf{c}$ to get input $\mathbf{x} \in \mathbb{R}^n$ to the channel given by

$$x_i := \sqrt{P}(2c_i - 1) \tag{3}$$

where $\sqrt{P}$ is the power constraint on the channel. As c_i is binary, $x_i = \pm\sqrt{P}$. The output of the channel $\mathbf{y}$ is conditionally distributed given input $\mathbf{x}$ by the channel pmf (or pdf) $W(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} W(y_i|x_i)$ as the channel is memoryless. When the decoder receives $\mathbf{y}$, it decodes the maximum aposteriori probability (MAP) estimate of $\mathbf{x}$, assuming equiprobable messages, on a symbol by symbol basis is given by

$$\hat{x}_i(\mathbf{y}) = \arg\max_{x_i\in\{0,1\}} \sum_{x_{-i}} \prod_{j=1}^{n} W(y_j|x_j)P(\mathbf{x}) \tag{4}$$

where $\sum_{x_{-i}}$ means summing over all components of $\mathbf{x}$ except x_i.

Equation (4) has a sum-product form, which is often simplified using techniques from factor graphs (see [8], [14]). However, for efficient computation, we prefer working with log likelihood ratios (LLR), which are defined for distributions u that are conditioned on channel inputs as

$$L(u) = \log\frac{u(\sqrt{P})}{u(-\sqrt{P})}, \tag{5}$$

rather than channel pmf (or pdf). For additive white Gaussian (AWGN) channels, this leads to a significant speedup as

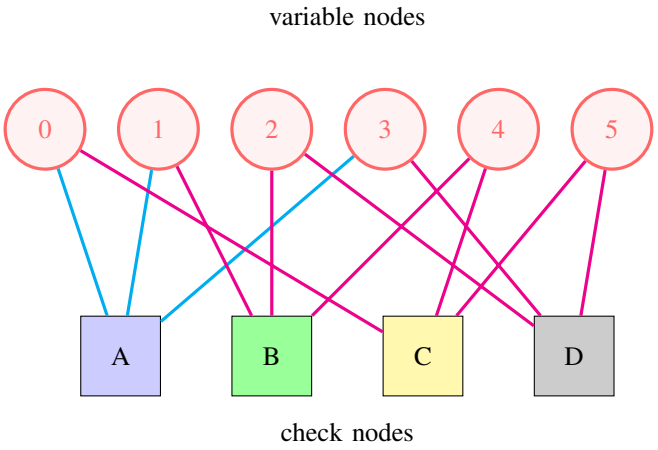

Fig. 1. A Tanner graph for the parity check matrix given in (1)

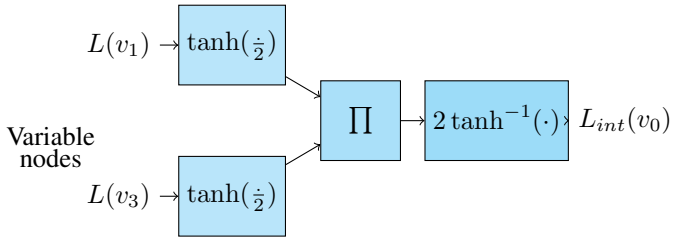

Fig. 2. Block diagram of existing soft decoder implementation of check node update.

computing an exponential (from the Gaussian pdf) has now reduced to working with a simple polynomial.

C. BP updates

A Tanner graph for an LDPC code as illustrated in Fig. 1, involves a bipartite graph consisting variable nodes (based on the codeword) on one end and check nodes on the other (for parity check). The belief propagation algorithm involves passing messages from variable nodes to check nodes and vice-versa. Let's concentrate on the message passed to check node A, refer to Fig 1. Check node 0 sends LLR information $L(v_0)$, similarly node 1 and 3 sends LLR information $L(v_1)$ and $L(v_3)$ respectively. Now the variable node 0,1 and 3 receive message from check node A. Fig 2 depicts what message variable node 0 receives from check node A, this message is called the intrinsic information for node 0 contributed by node 1 and 3. The total intrinsic LLR information received from all the check nodes at variable node 0 is given by the equation 6; where degree d is the number to check nodes attached to a variable node.

$$L_{intTotal}(v_0) = \sum_{i=0}^{d-1} L_{int}(v_{0i}) \tag{6}$$

Figure 2 can be generalized for all variable nodes using equation 7

$$L_{int}(v_j) = 2\tanh^{-1}\left[\prod_{\substack{k=0\\k\neq j}}^{z-1} \tanh\left(\frac{L(v_k)}{2}\right)\right]. \tag{7}$$

where $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ is the hyperbolic tangent function and $\tanh^{-1}(.)$ is its inverse. Here z is the number of connections each check node has. These functions are rather complex to implement. When the degree of the nodes, in this case determined by n_r and n_c of $\mathbf{H}$, increase, the computation complexity further increases. In the next section, we will discuss methods to mitigate these effects.

III. Proposed mechanisms for optimizing LDPC decoder

A. *Hyperbolic* $\tanh$ *and* $\tanh^{-1}$ *approximations.*

We try to approximate components of the intrinsic LLR function (equation written below) using linear piece-wise approximation and gradient descent. using equation 7

$$L_{int}(v_j) = 2\tanh^{-1}\left[\prod_{\substack{k=0\\k\neq j}}^{z-1} \tanh\left(\frac{L(v_k)}{2}\right)\right]. \tag{8}$$

CASE 1.

We first approximate $\tanh(x)$ of the above equation as using piece-wise linear functions i.e. $(ax + b)$. To learn the values of a and b we are using gradient descent. We first analyze $\tanh(x)$ and try to see how many different straight lines we can sufficiently use to approximate the above function. From the above diagram we decided to approximate the

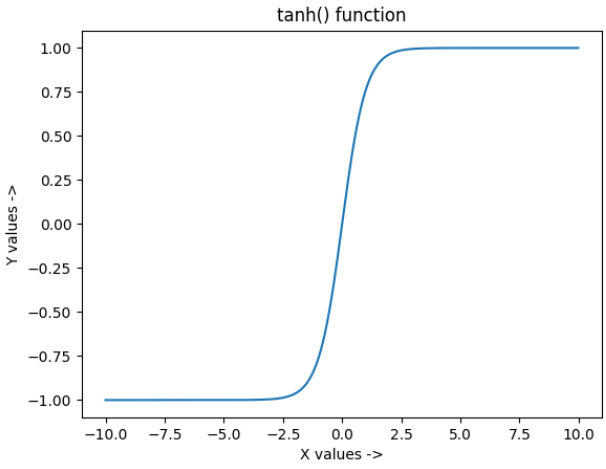

Fig. 3. A plot of $\tanh$ function.

$\tanh()$ using three straight lines. For $x < -1.2$ we have the first straight line. For $-1.2 <= x <= 1.2$ we have the second straight line and for $x >= 1.2$ we have the

third straight line. We estimate the coefficients of straight line a and b using gradient descent. For example in the range $x\epsilon[-1.2, 1.2]$ we generate N training examples $(x_1, \tanh(x_1)), (x_2, \tanh(x_2))..., (x_N, \tanh(x_N))$ s.t. $x_i \ \epsilon \ [-1.2, 1.2] \ \forall \ i$ from 1 to N. We use for following update rule to find the coefficients of the line $(ax + b)$.

$$a = a - \alpha \frac{\partial J(a,b)}{\partial a}$$
$$b = b - \alpha \frac{\partial J(a,b)}{\partial b} \tag{9}$$

Where α is the learning rate and should be sufficiently small but not too small. $J(a, b)$ is the mean squared error function. defined as,

$$J(a,b) = \frac{1}{N}\sum_{i=1}^{N}(\tanh(x_i) - ax_i - b)^2 \tag{10}$$

Where x_i is the ith training example. After running gradient descent following values of the coefficients were reached. **Figure 4**, shows how the piece-wise linear approximation

TABLE I
PIECE-WISE LINEAR COEFFICIENTS FOR $\tanh(x)$.

Index k	*Range*	a	b
1	$(-\infty, -1.2)$	0	-1
2	$[-1.2, 1.2)$	0.8435	0.0003
3	$[1.2, \infty)$	0	1

(PL approximation) fits the $\tanh(x)$ function. We plug the

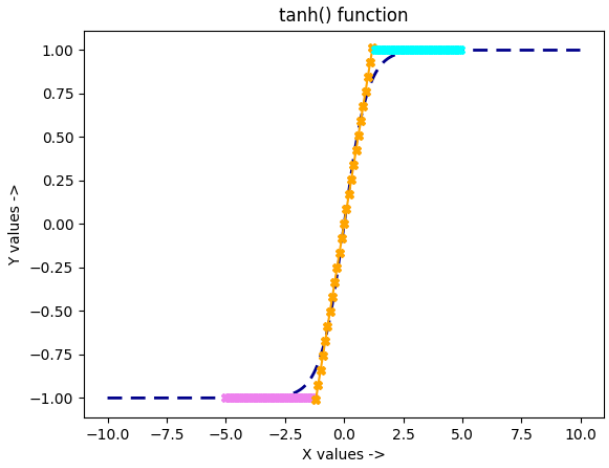

Fig. 4. PL approximation of $\tanh(x)$ function.

above piece-wise linear approximation in (7).

CASE 2.
Similarly, we also use the method described in CASE 1 to approximate $\tanh^{-1}(x)$. In this method we use 5 different straight lines to approximate the $\tanh^{-1}(x)$ function. More or less number of straight lines can be used depending on the required accuracy of the approximation. The results i.e. the coefficients of the straight lines are shown in TABLE II and visual representation of the approximation is depicted in Fig 5.

TABLE II
PIECE-WISE LINEAR COEFFICIENTS FOR $\tanh^{-1}(x)$.

Index k	*Range*	a	b
1	$(-1, -0.85)$	6.944	4.712
2	$[-0.85, -0.75)$	1.777	0.330
3	$[-0.75, 0.75)$	1.231	0.003
4	$[0.75, 0.85)$	1.815	-0.334
5	$[0.85, 1)$	8.347	-5.884

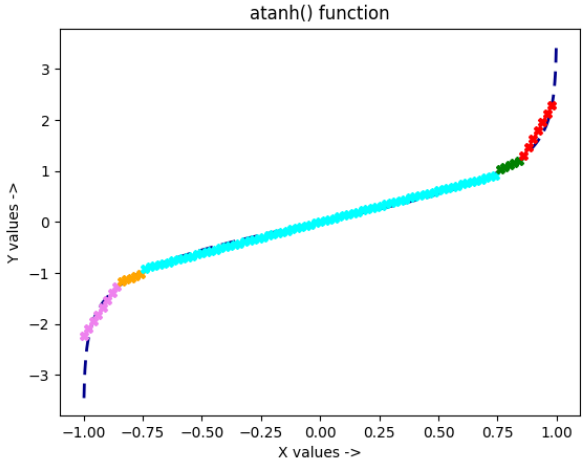

Fig. 5. PL approximation of $\tanh^{-1}(x)$ function.

CASE 3.
There is a requirement for LDPC decoder to be very fast. To make the decoding faster we combined approximations of CASE 1 and CASE 2. Results for this method are presented in the section **RESULTS AND COMPARISONS**. In **Figure 9** and **Figure 10** see *tnh_atnh_approx* in BLER v/s SNR graph.

CASE 4.
Here we explore other approximations of $\tanh^{-1}(x)$ which are not as fast as the previous methods but are slightly more accurate.

1) Polynomial Approximations: Instead of using piece-wise linear functions for approximating $\tanh^{-1}(x)$ we can use polynomial function $f(x)$.

$$f(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$

Higher order polynomials can be used to get more accuracy in lieu of time complexity. Using gradient descent the optimal values of the coefficients were discovered by reducing the loss function given by equation (11) using the update rules given by equation (12).

$$J(a_5, a_4, ..., a_0) = \frac{1}{N}\sum_{i=1}^{N}(\tanh^{-1}(x_i) - f(x))^2 \tag{11}$$

$$a_i = a_i - \alpha \frac{\partial J(a_5, a_4, ..., a_0)}{\partial a_i}$$
$$\forall i \epsilon \{5, 4, 3, 2, 1, 0\} \quad (12)$$

The values of the coefficients are given in the table below. A pictorial representation of the above approximation is shown in Fig. 6.

Coeff.	a_5	a_4	a_3	a_2	a_1	a_0
Values	3.91	0	-2.72	0	1.48	0

Since, $\tanh^{-1}(x)$ is an odd function the values of the coefficients a_4,a_2 and a_0 are zero. The mean squared error of the above model was calculated to be approximately ≈ 0.008.

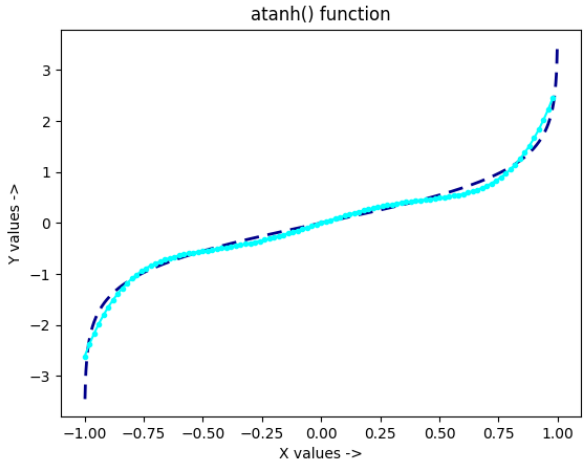

Fig. 6. Polynomial approximation of $\tanh^- 1(x)$ function.

2) Neural Network Approximations: The Neural Network model shown in Fig. 7. was used to approximate $\tanh^- 1(x)$ function. We generate training examples by taking $(N = 5000)$ points denoted by x and $y = \tanh^- 1(x)$. To the Neural Network represented by Fig. 7. we feed (x, x^2, x^3) which constitutes the input layer. Even power terms of x can be removed from the input layers since we are approximating an odd function. It was found that using $\{x, x^2 \ldots x^N\}$ in the

Input layer	Hidden Layer Id	No Of Neurons	Activation Function	Method	Loss Function
x,x^2,x^3	layer1	4	linear	Learning Rate - 0.1 & Adaptive gradient descent	Mean Squared Error
	layer2	2	linear		
	layer3	1	linear		

Fig. 7. Neural Network model for approximating $\tanh^- 1(x)$ function.

input layers to train the Neural Network defined above is better than using N piecewise linear functions for approximating $\tanh^- 1(x)$. The mean squared error on randomly generate 500 test examples were found to be around 0.01. It was found that using higher odd degree values of x in the input layer of the Neural Network reduced the MSE error significantly.

B. *Modified min-sum approximation.*

Evaluating **Equation** (7) is time consuming so researchers have come up with min-sum approximation of the above expression .

$$L_{int}(v_j) = \left[\prod_{\substack{k=0 \\ k \neq j}}^{z} sgn(L(v_k))\right] \left[\min_{\substack{\forall i \epsilon [0,z] \\ i \neq j}} |L(v_i)|\right] \quad (13)$$

A more accurate approximation of (7), is depicted by (13).

$$L_{int}(v_j) = \left[\prod_{\substack{k=0 \\ k \neq j}}^{z} sgn(L(v_k))\right] \left[a_1 \min_{\substack{\forall i \epsilon [0,z] \\ i \neq j}} |L(v_i)| + a_2\right] \quad (14)$$

We work on the equation approximation where $sgn(x)$ is the sign of x, and a_1, a_2 are some coefficients. These coefficients can be varied so as to get the best value for $L_{int}(v_j)$. The novelty in this method is that using linear regression, we can train and find the optimum values of the coefficients. Our training data must exactly mimic the real test data. To get good results we have trained a_1 and a_2 corresponding to the standard deviation of received signal or the noise, the coefficient value will also depend on the number of $\tanh(x)$ functions in equation 7 i.e. the value of z. For $z = 8$ we have calculated the value of a_1 and a_2 for different noise levels in TABLE III In **RESULTS AND COMPARISONS** subsection

TABLE III
OPTIMUM PARAMETER VALUES FOR DIFFERENT NOISE STANDARD DEVIATIONS.

Std. deviation σ	a_1	a_2
0.1	1.5014	-0.0022
0.2	1.1742	-0.0556
0.3	0.8513	-0.0464
0.4	0.6218	-0.0204
0.5	0.4713	-0.0087
0.6	0.3703	-0.0255
0.7	0.2715	-0.0022
0.8	0.2611	-0.0126
0.9	0.2597	0.0063

the BLER v/s SNR graph shows the performance of the above method shown as a blue line labelled as ML-method.

C. *Replacing Decoder with Neural Network*

In this method we replace the entire decoder block with a Neural Network block. To generate the input and output sets we first take M valid codewords which satisfies the parity check equation for a particular H matrix, these M valid codewords form our output set which is denoted by Y, to get the input set we use BPSK modulation and add white Gaussian noise of mean$(\mu = 0)$ and standard deviation$(\sigma = 0)$ to Y to generate input set X. The dimensions of X and Y is (M,N). Where M is number of training examples and

Input layer	Hidden Layer Id	No Of Neurons	Activation Function	Method	Loss Function
N-LLRs corresponding to N-codeword bits	layer1	4*N	tanh	Adaptive gradient descent	Binary Crossentropy
	layer2	2*N	linear		
	layer3	2*N	tanh		
	layer4	N	sigmoid		

Fig. 8. Neural Network model for decoder(Code-word Size=32).

N is the codeword size. (X, Y) was randomly split into training set (Xtr, Ytr) and test set $(Xtest, Ytest)$ in the ratio of $9:1$. The training set was fed to the Neural Network shown in Fig 8. We have created different Neural Network models based on different noise's standard deviation(σ) i.e. $\{NN_{\sigma 1}, NN_{\sigma 2}...NN_{\sigma L}\}$ where L is the maximum number of noise levels identified in the model. Based on the closest noise level we feed the input LLR to the appropriate Neural Network. It was found that higher codewords need Neural Network with more hidden layers, one could also use more hidden layers for high noise σ but we have used the same Neural Network architecture for all noise σ. Refer to Fig. 11 in the results section to see how our model compares to the original LDPC implementation for small codeword sizes.

IV. Results and Comparisons

We describe each section of the LDPC encoder and decoder design that was considered in our simulations. In all simulations, an AWGN channel with zero mean and variance σ^2 is chosen as the BIMC along with BPSK modulation scheme. The value of input power constraint P is taken as unity and σ^2 will be calculated based on signal to noise ratio (SNR) values. The LDPC matrices for various parameters given below were generated using Gallager's method [4]. Encoding of messages was carried out using (2).

A. PL-Approximation

In this procedure named piece-wise linear(PL) approximation, we use TABLE I and TABLE II for approximating $tanh(x)$ and $tanh^{-}1(x)$ respectively. The working of this procedure is explained in CASE 1 and 2 of section III.

B. ML-Approximation

The ML approximation method, unlike the PL one has coefficients that vary with system parameters like SNR. For a given SNR, we design a training set that is constructed using input LLR values $L(v_i)$ generated at random, and the corresponding output LLR computed via (7). A simple linear regression will then give us the optimum coefficients. A set of coefficients for specified noise standard deviation is provided in Table III.

C. Replacing Decoder with Neural Network

In order to generate BLER V/S SNR graph for Neural Network Decoder we took the codeword size $(N = 32)$ and message bit size $(K = 22)$. BLER was calculated on test data using the appropriate Neural Network model(as explained earlier different Neural Network models are used based on the noise level).

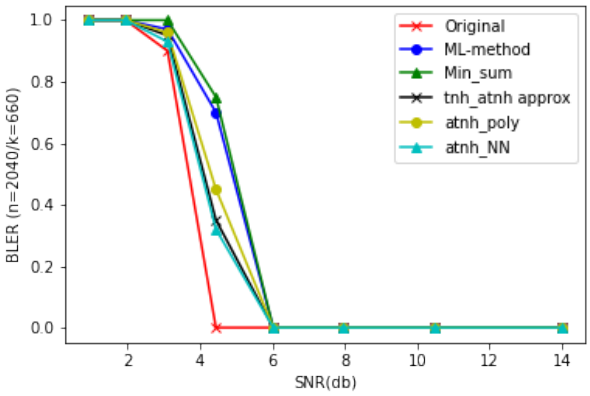

Fig. 9. Block error rate comparison of proposed methods i.e. ML_approx and PL_approx(or tnh_atnh approx) versus known methods like min-max approximation and original method for $n = 2040$, $k = 660$.

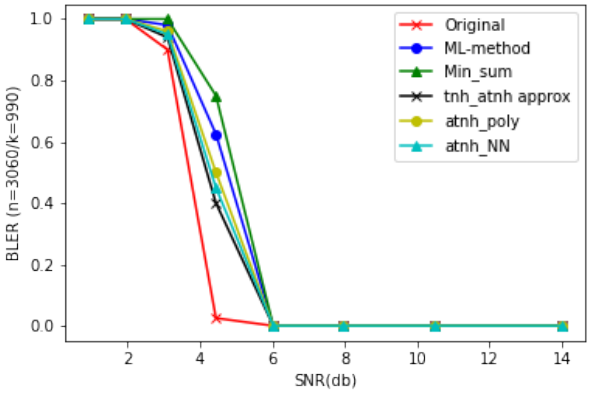

Fig. 10. Block error rate comparison of proposed methods i.e. ML_approx and PL_approx(or tnh_atnh approx) versus known methods like min-max approximation and original method for $n = 3060$, $k = 990$.

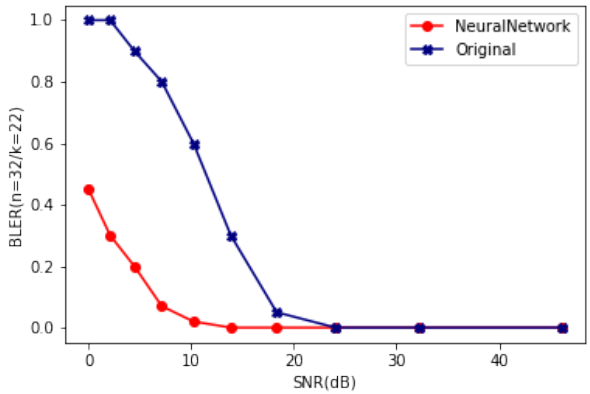

Fig. 11. Block error rate comparison of proposed Neural Network method versus known original method(no approximations) for $n = 32$, $k = 22$.

D. Comparison with existing methods

From Fig. 9 and 10, we see that the PL-Approximation has near matching block error rate (BLER) performance with the original implementation and is significantly better than the min-sum method typically used. From Fig. 11 we can conclude that our Neural Network decoder model performs significantly better than LDPC decoder for the small codeword sizes.

V. Conclusion

We approximated the complex LLR update soft decoder expression (7) with piece-wise linear and machine learning approximations. We have shown via simulations that the computation complexity and speed are improved with these methods as opposed to direct implementation. Our Neural Network decoder's performance is also significantly better than normal LDPC implementation. With the advent of vRAN and O-RAN, these methods are now practically implementable and may be considered for replacement of existing decoding modules.

References

[1] E. Dahlman, G. Mildh, S. Parkvall, J. Peisa, J. Sachs, Y. Selén, and J. Sköld, "5G wireless access: requirements and realization," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 42–47, 2014.

[2] Y. Niu, Y. Li, D. Jin, L. Su, and A. V. Vasilakos, "A survey of millimeter wave communications (mmWave) for 5G: opportunities and challenges," *Wireless networks*, vol. 21, no. 8, pp. 2657–2676, 2015.

[3] (2019) Operator Defined Next Generation RAN. [Online]. Available: https://www.o-ran.org/

[4] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, January 1962.

[5] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[6] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on information theory*, vol. 27, no. 5, pp. 533–547, 1981.

[7] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Transactions on information theory*, vol. 47, no. 2, pp. 638–656, 2001.

[8] J. Chen and M. P. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Transactions on communications*, vol. 50, no. 3, pp. 406–414, 2002.

[9] J. Chen, A. Dholakia, E. Eleftheriou, M. P. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of ldpc codes," *IEEE transactions on communications*, vol. 53, no. 8, pp. 1288–1299, 2005.

[10] S. Lin and D. J. Costello, *Error control coding*. Prentice hall, 2001, vol. 2.

[11] Z. Li, L. Chen, L. Zeng, S. Lin, and W. H. Fong, "Efficient encoding of quasi-cyclic low-density parity-check codes," *IEEE Transactions on Communications*, vol. 54, no. 1, pp. 71–81, 2006.

[12] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on information theory*, vol. 47, no. 2, pp. 599–618, 2001.

[13] A. Ashikhmin, G. Kramer, and S. ten Brink, "Extrinsic information transfer functions: model and erasure channel properties," *IEEE Transactions on Information Theory*, vol. 50, no. 11, pp. 2657–2673, 2004.

[14] H.-A. Loeliger, "An introduction to factor graphs," *IEEE Signal Processing Magazine*, vol. 21, no. 1, pp. 28–41, 2004.