Near Maximum Likelihood Decoding with Deep Learning

Article · January 2018

CITATIONS
4

READS
248

5 authors, including:

David Burshtein
Tel Aviv University
99 PUBLICATIONS **2,956** CITATIONS
SEE PROFILE

Yair Be'ery
Tel Aviv University
80 PUBLICATIONS **1,320** CITATIONS
SEE PROFILE

Near Maximum Likelihood Decoding with Deep Learning

Eliya Nachmani, Yaron Bachar, Elad Marciano, David Burshtein and Yair Be'ery
School of Electrical Engineering, Tel-Aviv University, Israel
Emails: enk100@gmail.com, yaronbac@gmail.com, eladmarc@gmail.com, burstyn@eng.tau.ac.il, ybeery@eng.tau.ac.il

***Abstract*—A novel and efficient neural decoder algorithm is proposed. The proposed decoder is based on the neural Belief Propagation algorithm and the Automorphism Group. By combining neural belief propagation with permutations from the Automorphism Group we achieve near maximum likelihood performance for High Density Parity Check codes. Moreover, the proposed decoder significantly improves the decoding complexity, compared to our earlier work on the topic. We also investigate the training process and show how it can be accelerated. Simulations of the hessian and the condition number show why the learning process is accelerated. We demonstrate the decoding algorithm for various linear block codes of length up to 63 bits.**

I. INTRODUCTION

In the last few years Deep Learning methods were applied to communication systems, for example in [1]–[8]. Furthermore, Deep Neural decoders is a new approach for decoding linear block codes. In [9]–[12] it has been shown that deep neural decoders can improve the existing belief propagation methods for decoding high density parity check codes (HDPCs). Other methods for using deep learning to decode error correcting codes were proposed in [13]–[15]. In this work we combine the deep recurrent neural decoder of [10] with permutations from the Automorphism Group as defined in [16]. The combined architecture is defined by $I_{permutations}$ blocks, each of which contains I_{BP} iterations of neural belief propagation followed by permutation. We show that this architecture achieves near maximum-likelihood performance for various BCH codes of up to 63 bits long with significantly lower complexity then the corresponding mRRD decoder [17]. We also investigate the training process of the deep neural decoder and show how the learning can be accelerated by adding penalties to the loss function. We argue that this penalties transform the manifold of the loss function into an isotropic manifold which is easy to optimize. Simulations of the Hessian matrix of the loss function support this claim.

II. THE NEURAL BELIEF PROPAGATION ALGORITHM

We start with a brief description of the deep neural network proposed in [9], [10]. The deep neural decoder is a message passing algorithm parameterized as a deep neural network. The input to the neural network is the set of LLR values, $v = 1, 2, \ldots, N$,

$$l_v = \log \frac{\Pr\left(C_v = 1 | y_v\right)}{\Pr\left(C_v = 0 | y_v\right)}$$

where N is the block length of the code, y_v is the channel output corresponding to the vth codebit. The neural decoder consists of pairs of odd and even layers. For odd i layer,

$$x_{i,e=(v,c)} = \\ = \tanh\left(\frac{1}{2}\left(l_v + \sum_{e'=(c',v),\ c' \neq c} w_{e,e'} x_{i-1,e'}\right)\right) \quad (1)$$

for even i layer,

$$x_{i,e=(c,v)} = 2\tanh^{-1}\left(\prod_{e'=(v',c),\ v' \neq v} x_{i-1,e'}\right) \quad (2)$$

and for output layer,

$$o_v = \sigma\left(l_v + \sum_{e'=(c',v)} \tilde{w}_{v,e'} x_{2L,e'}\right) \quad (3)$$

where $\sigma(x) \equiv (1 + e^{-x})^{-1}$ is a sigmoid function. Please note that equations (1),(2) define recurrent neural network, as the learnable weights $w_{e,e'}, \tilde{w}_{v,e'}$ are tied.

III. THE PROPOSED DEEP NEURAL NETWORK DECODER

A. Architecture

The proposed neural network is composed of $I_{permutations}$ blocks. Each block contains I_{BP} layers of neural belief propagation, which are described below. Between each two successive blocks, we apply a permutation from the automorphism group. Lastly, we apply the corresponding inverse permutation, to obtain the decoded codeword. The proposed architecture is illustrated in Figure 1.

We re-parameterized the deep neural network decoder from section II. In the j-th block, I_{BP} iterations of neural belief-propagation are performed as follows:

For each variable node in the i-th layer,

$$x^j_{i,e=(v,c)} = \tanh\left(\frac{1}{2}(o^j_{i-1,v} - x^j_{i-1,e=(c,v)})\right) \quad (4)$$

For each check node in the i-th layer,

$$x^j_{i,e=(c,v)} = 2\tanh^{-1}\left(\prod_{e'=(v',c),\ v' \neq v} x^j_{i-1,e'}\right) \quad (5)$$

For mid-output node in the i-th layer,

$$o_{i,v}^{j} = o_{0,v}^{j} + \sum_{e'=(c',v)} w_{e'} x_{i,e'}^{j} \tag{6}$$

The output of the j-th block:

$$c_{v}^{j} = \pi_{j}(o_{i=I_{BP},v}^{j}) \tag{7}$$

The initialization:

$$o_{0,v}^{j} = \begin{cases} l_v & ,j=0 \\ c_v^{j-1} & ,j>0 \end{cases} \tag{8}$$

$$x_{0,e}^{j} = 0 \tag{9}$$

After each BP block, an appropriate inverse permutation is applied:

$$d_{v}^{j} = (\pi_{1}^{-1} \cdot \pi_{2}^{-1} \cdot ... \cdot \pi_{j}^{-1}) c_{v}^{j} \tag{10}$$

Note that the neural weights $w_{e'}$ are tied along the neural network graph. Also note, that the new parametrization of the neural belief propagation decoder was easier to optimize, and converged faster to a better performance.

As in [9]–[12] the proposed architecture preserves the symmetry conditions, therefore we can train the neural network with noisy versions of a single codeword.

Also note, that in order to be consistent with the BP algorithm, one needs to multiply $x_{i-1,e=(c,v)}^{j}$ in (4) by w_e. However, this multiplication did not have any significant influence on the results obtained.

B. Loss function

The loss of the neural network is composed of three constituents:

- A multi-loss cross-entropy between $\tilde{d}_{v}^{j} \equiv \sigma(d_{v}^{j})$ and the correct codeword y_v. This is a loss term concerned with the output of the BP blocks:

$$L_{1}^{j} = -\frac{1}{N} \sum_{v=1}^{N} y_v \log(\tilde{d}_{v}^{j}) + (1-y_v) \log(1-\tilde{d}_{v}^{j}) \tag{11}$$

- A sub multi-loss cross-entropy between $\tilde{o}_{i,v}^{j} \equiv \sigma(o_{i,v}^{j})$ and the correct codeword y_v. This term involves inner-BP marginalizations:

$$L_{2}^{j,i} = -\frac{1}{N} \sum_{v=1}^{N} y_v \log(o_{i,v}^{j}) + (1-y_v) \log(1-o_{i,v}^{j}) \tag{12}$$

- l_2-norm of the weights w_v, $w_{v,e'}$:

$$L_3 = \sum_{v} \|w_v\|^2 + \sum_{v,e} \|w_{v,e}\|^2 \tag{13}$$

The total loss is:

$$L = \sum_{j} (L_{1}^{j} + \lambda \cdot L_3) + \sum_{j,i} L_{2}^{j,i} \tag{14}$$

IV. EXPERIMENTS

A. Neural Network Training and Dataset

We implemented the proposed neural network in TensorFlow framework. The neural network was optimized with RMSPROP [18]. As in [9]–[12], the dataset consisted of the zero codeword and an AWGN channel. We used the cycle reduced parity check matrix from [19]. Due to large number of layers in our network, and the fact that the network is a recurrent neural network, gradient clipping was applied to avoid gradient exploding throughout the learning process. Clipping threshold of $c_{grad} = 0.1$ was used. The l_2-Loss term was added with a factor of λ. Note that we use the three terms L_1, L_2, L_3 of the loss for training. The weights were constrained to have non-negative values. In all of our experiments no overfitting was observed.

The architecture was tested on BCH codes. Their automorphism group is described in detail in [20]. The permutations were chosen randomly using the product-replacement algorithm [19], which has the N_{pr} and K_{pr} parameters. N_{pr} is the size of the group of permutations the algorithm builds, and K_{pr} is the initial number of iterations, used to build this permutations-reservoir. In Table I we provide details about the parameters configurations of the network.

B. BCH(63,45)

Batch size was set to 160, with 20 examples per SNR. The SNR varied from $1dB$ to $8dB$ in the training process, and from $1dB$ to $5dB$ in the validation process. The neural network comprises $I_{permutations} = 50$ permutations, and each block contains $I_{BP} = 2$ BP iterations. A total of 100 BP iterations correspond to a deep neural network with 200 layers.

C. BCH(63,36)

Batch size was set to 120, with 30 examples per SNR. The SNR varied from $1dB$ to $6dB$ in the training process, and from $3dB$ to $4.5dB$ in the validation process. The neural network comprised $I_{permutations} = 300$ permutation, and each block contains $I_{BP} = 2$ belief propagation iteration. This configuration represents 600 Belief Propagation iterations which correspond to deep neural network with 1200 layers.

	Parameter	BCH(63,45)	BCH(63,36)
BP	I_{BP}	2	2
	llr clip	15	15
RRD	$I_{Permutatoins}$	50	300
	N_{pr}	20	1000
	K_{pr}	60	4000
Neural Network	learning rate	1e-3	1e-3
	batch size	160	120
	batch size / snr	20	30
	SNR range	1-8dB	1-6dB
	λ	100	10^{12}
	gradient clipping	0.1	0.1
	network depth	200	1200

TABLE I: Parameter Configuration of the Model

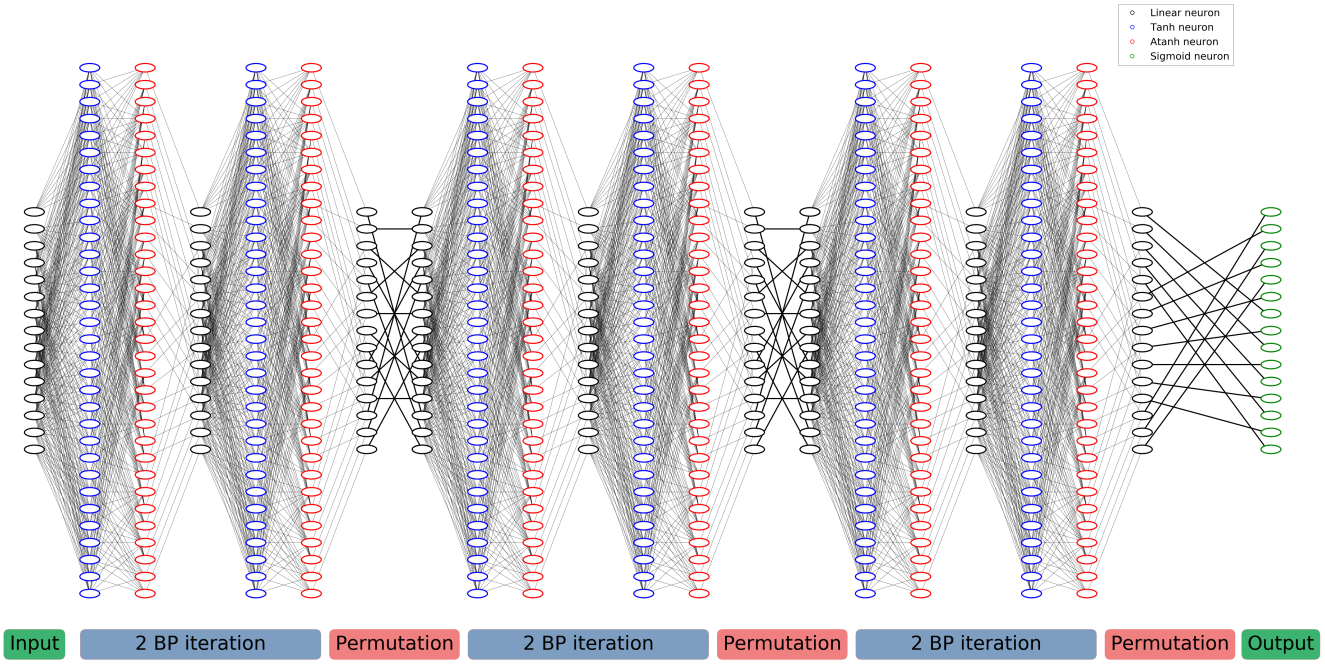

Fig. 1: Deep Neural Network Architecture For BCH(15,11) with 3 permutations and 2 belief propagation iterations for each permutation. The permutations have bold lines. The self message $o_{i,v}^{j}$ was removed from the diagram for a cleaner view.

D. Results

In the following figures, "Perm-RNN-i-j-k" denotes our proposed decoder, with i parallel branches, j permutations and k BP iterations between two consecutive permutations; "mRRD-i" denotes the classical mRRD decoder with i branches; and "mRRD-RNN-i-j-k" denotes the mRRD-RNN decoder with i branches, j blocks of BP, each with k iterations.

In Figure 2 we provide the bit-error-rate for $BCH(63, 45)$ code for our proposed decoder. The maximum-likelihood estimate was obtained by the OSD algorithm [21]. We observe near maximum likelihood performance with our proposed decoder, with a gap of up to 0.2dB to ML. The runtime of the proposed neural decoder is lower than OSD's when SNR is bigger than 3.8dB, as shown in figure 4. In Figures 5 and 6 we provide the bit-error-rate and the running time for $BCH(63, 36)$ code for our proposed decoder. The maximum-likelihood estimate was obtained by the 2nd order OSD algorithm [21], and the mRRD performance was obtained using 10-parallel mRRD decoder [17]. We have a gap of 0.25-0.5dB to achieve maximum likelihood performance with our proposed decoder.

Note, that the overall decoding time of our decoder is substantially smaller than the mRRD's decoding time for the (63,36) code, with a factor of up to 3.5. In addition, only one neural decoder was needed to match the performance 10-parallel mRRD decoder. Also note, that OSD's main disadvantage of parallel implementation is not encountered in the neural decoder.

In Figure 3 we provide the learning curve for $BCH(63, 45)$ code. The learning rate was constant during the training process, yet the loss significantly drops at some stage of the training. For training without l_2-norm, the drop occurs in epoch 265, and most of the improvement occurs at the same time. Training with l_2-norm accelerated the learning process: the loss dropped at epoch 8, as if the training process was accelerated by factor 33. We will investigate and discuss the dropping phenomenon and the l_2-acceleration at section V.

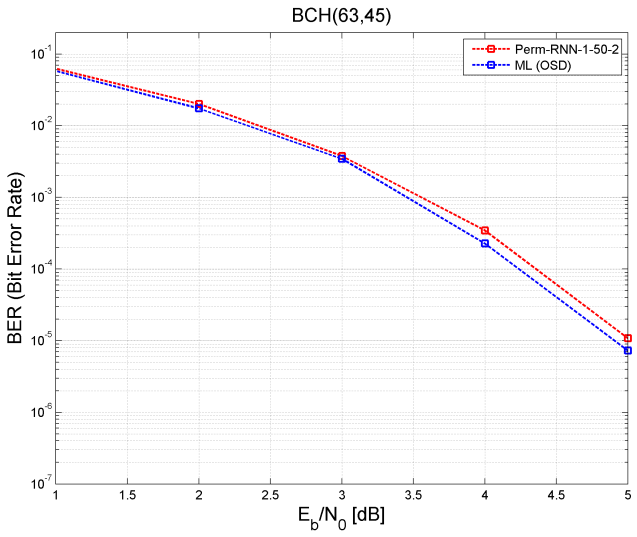

Fig. 2: BER results for BCH(63,45) code

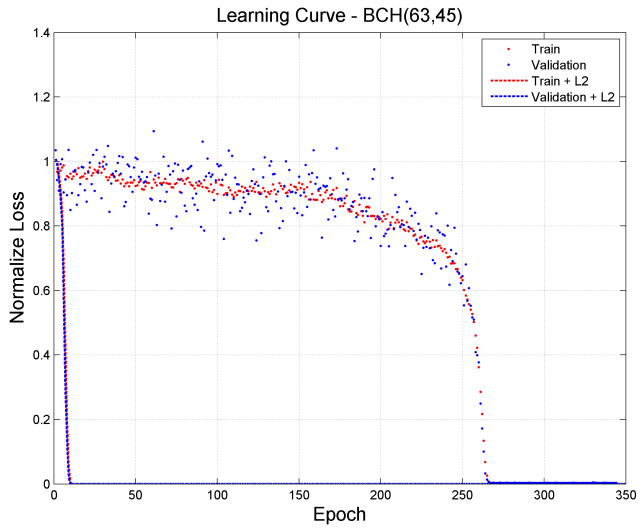

Fig. 3: Learning Curve for BCH(63,45) code

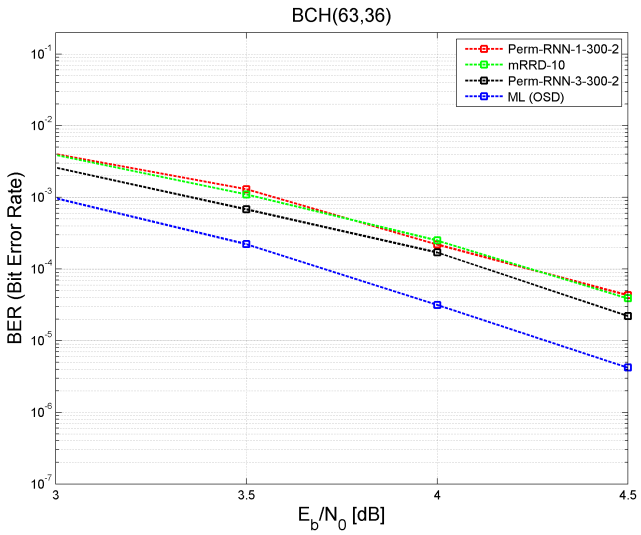

Fig. 5: BER results for BCH(63,36) code

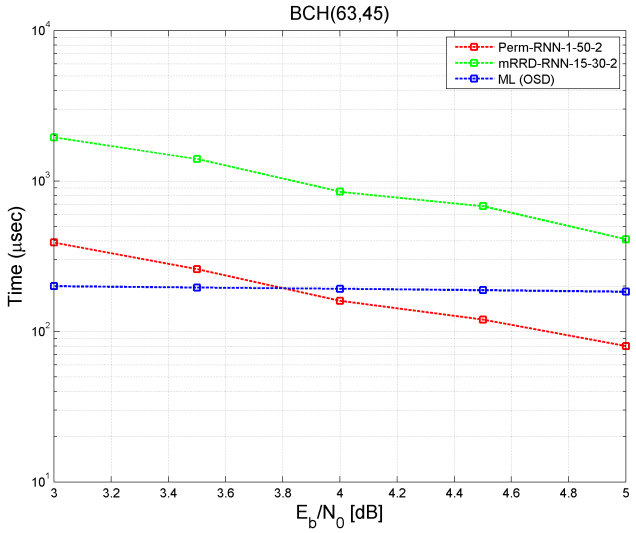

Fig. 4: Running time comparison for BCH(63,45) code

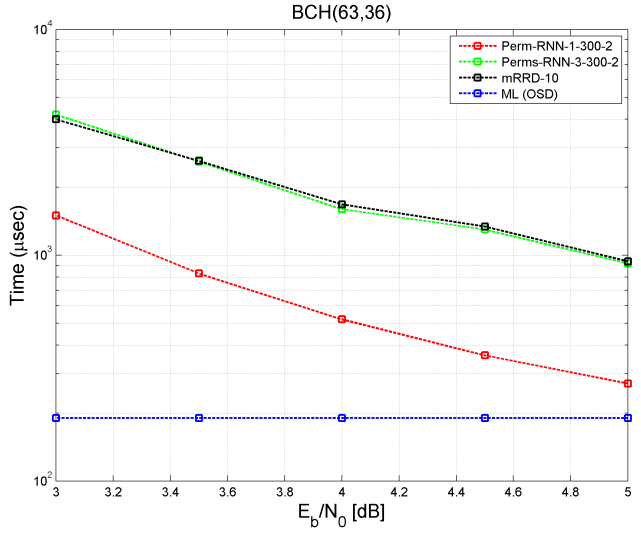

Fig. 6: Running time comparison for BCH(63,36) code

V. TRAINING ACCELERATION

As introduced in the previous section, during the training the loss drops significantly. This phenomenon usually occurs while training very deep neural networks. Note that our proposed network for $BCH(63, 45)$ contained 200 layers. As shown in [22], this phenomenon can be explained by the existence of saddle points in the loss-surface of the network.

A. Hessian simulation

To further investigate the phenomenon of the significant loss-drop and the l_2 acceleration, we computed the Hessian matrix of the deep neural decoder. Since the Hessian calculation demands high resources, we investigated the training process of a similar and smaller code, $BCH(31, 16)$. As shown in Figures 7 and 8, the training process of the $BCH(31, 16)$ behaves in the same manner as the $BCH(63, 45)$ code.

The Hessian matrix was evaluated during the training process. We calculated the condition number and the distribution of the eigenvalues of the Hessian matrix. The setting for the $BCH(31, 16)$ code was: $I_{permutations} = 10$ permutations, $I_{BP} = 2$ BP-iterations, $c_{grad} = 0.1$, $\lambda = 100$.

Figures 7 and 9 demonstrate the significant loss-drop properties. Whereas in epochs 1-20 the loss and the BER do not improve significantly and the positive eigenvalues ratio is low, epoch 20-40 serves as a turning point: the loss and the BER decrease rapidly and the positive eigenvalues ratio increases at the same time.

Figures 8 and 9 further stress this matter: the positive eigenvalues ratio is high right from the beginning, and accordingly

the loss presents no initial-plateau to begin with. Put in other words, the Hessian rapidly becomes similar to a scaled identity matrix. The equivalent loss-surface is isotropic, which results in an accelerated learning process.

It is of no surprise that adding an l_2 term to the loss brings the Hessian closer to an identity matrix. Yet, the notable training acceleration and the performance improvement are a result of a gentle setting of parameters and the specific optimization problem discussed.

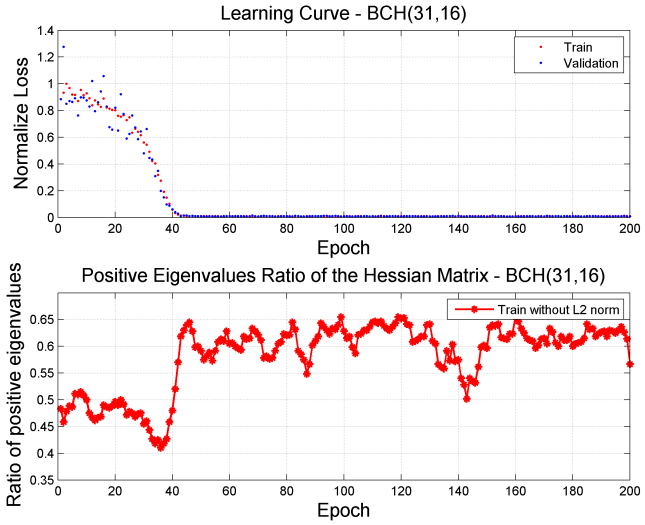

Fig. 7: Learning Curve and Positive Eigenvalues Ratio of the Hessian Matrix for BCH(31,16) For Training without l_2-norm

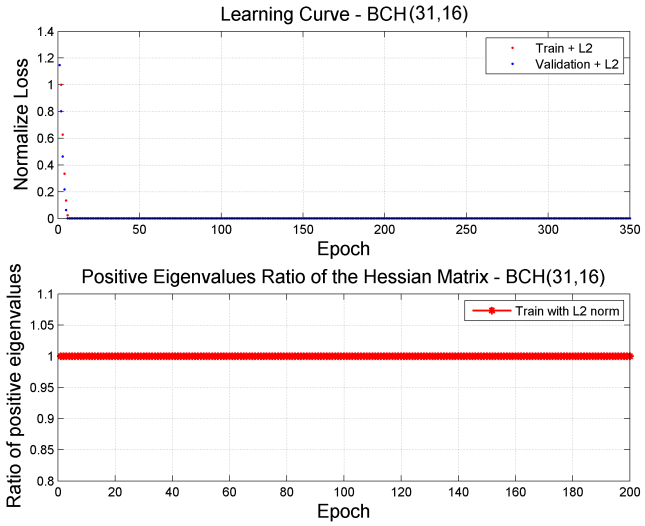

Fig. 8: Learning Curve and Positive Eigenvalues Ratio of the Hessian Matrix for BCH(31,16) For Training with l_2-norm

REFERENCES

[1] N. Farsad and A. Goldsmith, "Detection algorithms for communication systems using deep learning," *arXiv preprint arXiv:1705.08044*, 2017.

[2] T. J. O'Shea and J. Hoydis, "An introduction to machine learning communications systems," *arXiv preprint arXiv:1702.00832*, 2017.

[3] N. Samuel, T. Diskin, and A. Wiesel, "Deep mimo detection," *arXiv preprint arXiv:1706.01151*, 2017.

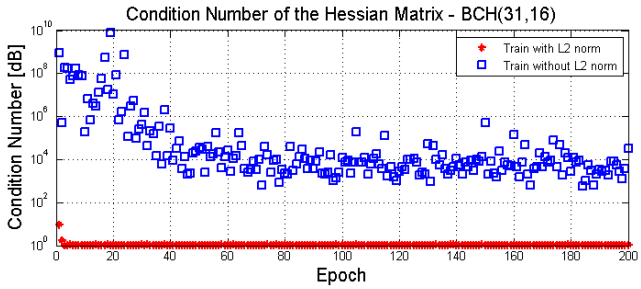

Fig. 9: Condition Number of the Hessian Matrix during training for BCH(31,16) code

[4] F. Liang, C. Shen, and F. Wu, "An iterative bp-cnn architecture for channel decoding," *arXiv preprint arXiv:1707.05697*, 2017.

[5] S. Dorner, S. Cammerer, J. Hoydis, and S. ten Brink, "Deep learning-based communication over the air," *arXiv preprint arXiv:1707.03384*, 2017.

[6] P. Shengliang, J. Hanyu, W. Huaxia, and Y. Yu-Dong, "Deep learning and its applications in communications systems - modulation classification," *Submitted to IEEE Communications Magazine*, 2017.

[7] Y. Hao, Y. L. Geoffrey, and F. J. Biing-Hwang, "Power of deep learning for channel estimation and signal detection in ofdm systems," *arXiv preprint arXiv:1708.08514*, 2017.

[8] H. Sihao and L. Haowen, "Fully optical spacecraft communications: Implementing an omnidirectional pv-cell receiver and 8mb/s led visible light downlink with deep learning error correction," *arXiv preprint arXiv:1709.03222*, 2017.

[9] 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.

[10] E. Nachmani, E. Marciano, D. Burshtein, and Y. Be'ery, "Rnn decoding of linear block codes," *arXiv preprint arXiv:1702.07560*, 2017.

[11] 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.

[12] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Beery, "Deep learning methods for improved decoding of linear codes," *IEEE Journal of Selected Topics In Signal Processing*, Feb. 2018.

[13] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," *accepted for CISS 2017, arXiv preprint arXiv:1701.07738*, 2017.

[14] S. Cammerer, T. Gruber, J. Hoydis, and S. ten Brink, "Scaling deep learning-based decoding of polar codes via partitioning," *arXiv preprint arXiv:1702.06901*, 2017.

[15] S. Krastanov and L. Jiang, "Deep neural network probabilistic decoder for stabilizer codes," *arXiv preprint arXiv:1705.09334*, 2017.

[16] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*. Elsevier, 1977.

[17] I. Dimnik and Y. Be'ery, "Improved random redundant iterative hdpc decoding," *IEEE Transactions on Communications*, vol. 57, no. 7, pp. 1982–1985, 2009.

[18] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, vol. 4, no. 2, 2012.

[19] T. R. Halford and K. M. Chugg, "Random redundant soft-in soft-out decoding of linear block codes," in *Information Theory, 2006 IEEE International Symposium on*. IEEE, 2006, pp. 2230–2234.

[20] C.-C. Lu and L. R. Welch, "On automorphism groups of binary primitive bch codes," in *Proc. IEEE Symposium on Information Theory*, June 1994, p. 1951.

[21] M. P. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Transactions on Information Theory*, vol. 41, no. 5, pp. 1379–1396, 1995.

[22] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in neural information processing systems*, 2014, pp. 2933–2941.