Learning to Decode Linear Codes Using Deep Learning

Eliya Nachmani[1], Yair Be'ery[2] and David Burshtein[3]

***Abstract*— A novel deep learning method for improving the belief propagation algorithm is proposed. The method generalizes the standard belief propagation algorithm by assigning weights to the edges of the Tanner graph. These edges are then trained using deep learning techniques. A well-known property of the belief propagation algorithm is the independence of the performance on the transmitted codeword. A crucial property of our new method is that our decoder preserved this property. Furthermore, this property allows us to learn only a single codeword instead of exponential number of codewords. Improvements over the belief propagation algorithm are demonstrated for various high density parity check codes.**

I. INTRODUCTION

In recent years deep learning methods have demonstrated significant improvements in various tasks. These methods outperform human-level object detection in some tasks [1], and achieve state-of-the-art results in machine translation [2] and speech processing [3]. Additionally, deep learning combined with reinforcement learning techniques was able to beat human champions in challenging games such as Go [4]. There are three different reasons for the outstanding results of Deep Learning models:

1) Powerful computing resources such as fast GPUs.
2) Utilizing efficiently large collections of datasets, e.g. ImageNet [5] for image processing.
3) Advanced academic research on training methods and network architectures [6], [7], [8], [9].

Error correcting codes for channel coding are used in order to enable reliable communications at rates close to the Shannon capacity. A well-known family of linear error correcting codes are the low-density parity-check (LDPC) codes [10]. LDPC codes achieve near Shannon channel capacity with the belief propagation (BP) decoding algorithm, but can typically do so for relatively large block lengths. For high density parity check (HDPC) codes [11], [12], [13], [14], such as common powerful algebraic codes, the BP algorithm obtains poor results compared to the maximum likelihood decoder [15]. In this work we focus on HDPC codes and demonstrate how the BP algorithm can be improved. The naive approach to the problem is to assume a neural network type decoder without restrictions, and train its weights using a dataset that contains a large amount of codewords. The training goal is to reconstruct the transmitted codeword from a noisy version after transmitting over the communication channel. Unfortunately, when using this approach our decoder is not given any side information regarding the structure of the code. In fact it is even not aware of the fact that the code is linear. Hence we are required to train the decoder using a huge collection of codewords from the code, and due to the exponential nature of the problem, this is infeasible, e.g., for a BCH(63,45) code we need a dataset of 2^{45} codewords. On top of that, the database needs to reflect the variability due to the noisy channel. In order to overcome this issue, our proposed approach is to assign weights to the edges of the Tanner graph that represent the given linear code, thus yielding a "soft" Tanner graph. These edges are trained using deep learning techniques. A well-known property of the BP algorithm is the independence of the performance on the transmitted codeword. A major ingredient in our new method is that this property is preserved by our decoder. Thus it is sufficient to use a single codeword for training the parameters of our decoder. We demonstrate improvements over BP for various high density parity check codes, including BCH(63,36), BCH(63,45), and BCH(127,106) .

[1]Eliya Nachmani is with the School of Electrical Engineering, Tel-Aviv University, enk100@gmail.com

[2]Yair Be'ery is with the School of Electrical Engineering, Tel-Aviv University, ybeery@eng.tau.ac.il

[3]David Burshtein is with the School of Electrical Engineering, Tel-Aviv University, burstyn@eng.tau.ac.il

II. THE BELIEF PROPAGATION ALGORITHM

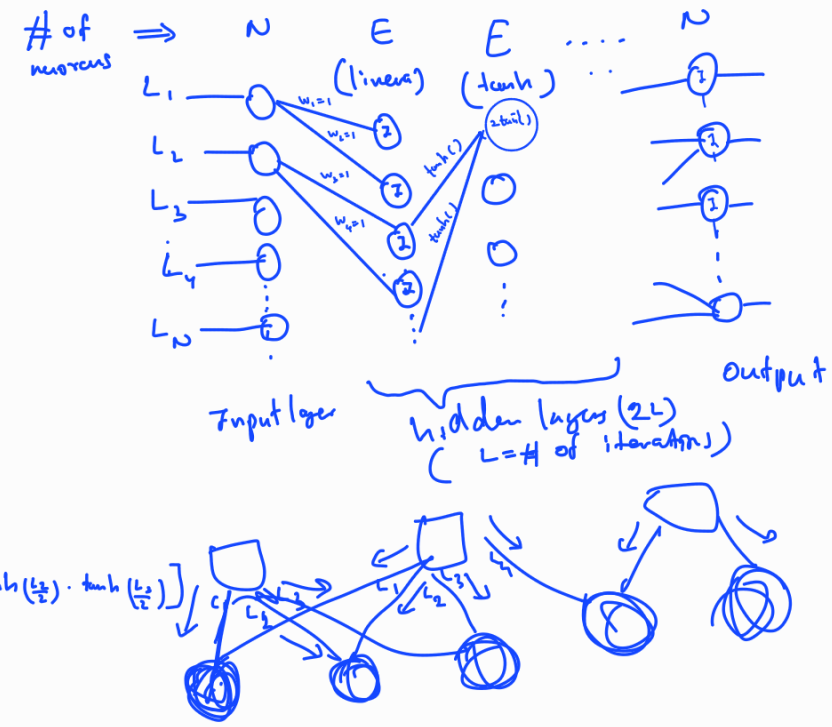
The renowned BP decoder [10], [16] can be constructed from the Tanner graph, which is a graphical representation of some parity check matrix that describes the code. In this algorithm, messages are transmitted over edges. Each edge calculates its outgoing message based on all incoming messages it receives over all its edges, except for the message received on the transmitting edge. We start by providing an alternative graphical representation to the BP algorithm with L full iterations when using parallel (flooding) scheduling. Our alternative representation is a trellis in which the nodes in the hidden layers correspond to edges in the Tanner graph. Denote by N, the code block length (i.e., the number of variable nodes in the Tanner graph), and by E, the number of edges in the Tanner graph. Then the input layer of our trellis representation of the BP decoder is a vector of size N, that consists of the log-likelihood ratios (LLRs) of the channel outputs. The LLR value of variable node v, $v = 1, 2, \ldots, N$, is given by

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

where y_v is the channel output corresponding to the vth codebit, C_v.

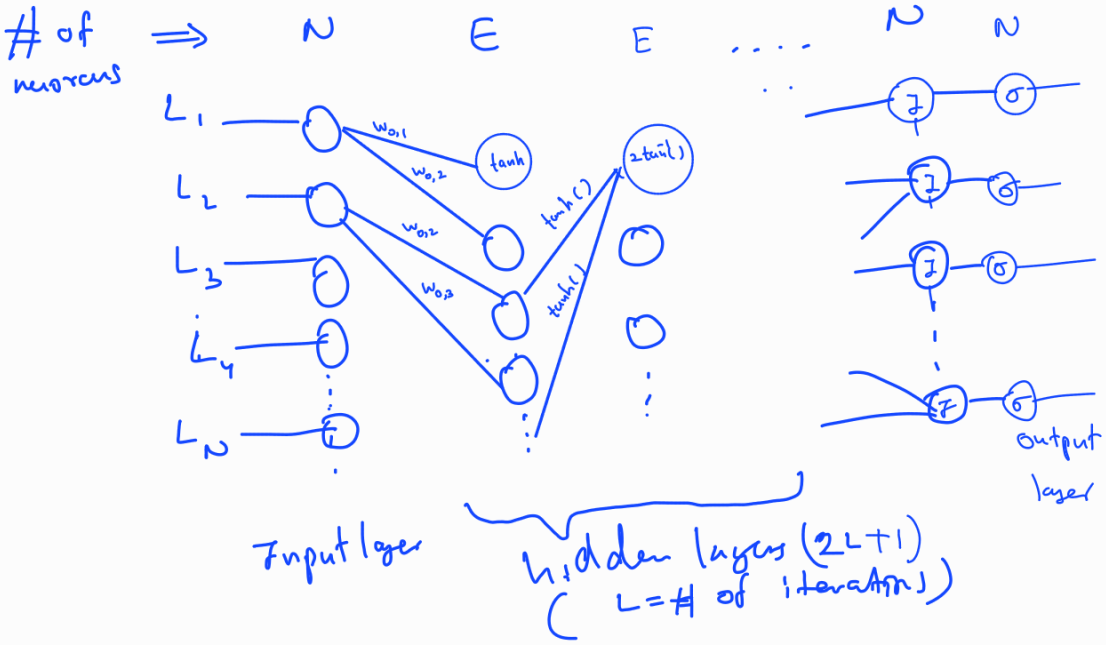
All the following layers in the trellis, except for the last one (i.e., all the hidden layers), have size E. For each hidden layer, each processing element in that layer is associated with the message transmitted over some edge in the Tanner graph. The last (output) layer of the trellis consists of N

NN Analogy



what these w's can do

* These weight allow the network to learn which messages are more reliable. (If some connections in the Tanner graph are noisy or from short cycles, the network can down-weight them.)



not fully connected (Sparse)

processing elements that output the final decoded codeword. Consider the ith hidden layer, $i = 1, 2, \ldots, 2L$. For odd (even, respectively) values of i, each processing element in this layer outputs the message transmitted by the BP decoder over the corresponding edge in the graph, from the associated variable (check) node to the associated check (variable) node. A processing element in the first hidden layer ($i = 1$), corresponding to the edge $e = (v, c)$, is connected to a single input node in the input layer: It is the variable node, v, associated with that edge. Now consider the ith ($i > 1$) hidden layer. For odd (even, respectively) values of i, the processing node corresponding to the edge $e = (v, c)$ is connected to all processing elements in layer $i-1$ associated with the edges $e' = (v, c')$ for $c' \neq c$ ($e' = (v', c)$ for $v' \neq v$, respectively). For odd i, a processing node in layer i, corresponding to the edge $e = (v, c)$, is also connected to the vth input node.

The BP messages transmitted over the trellis graph are the following. Consider hidden layer i, $i = 1, 2, \ldots, 2L$, and let $e = (v, c)$ be the index of some processing element in that layer. We denote by $x_{i,e}$, the output message of this processing element. For odd (even, respectively), i, this is the message produced by the BP algorithm after $\lfloor (i-1)/2 \rfloor$ iterations, from variable to check (check to variable) node.

For odd i and $e = (v, c)$ we have (recall that the self LLR message of v is l_v),

$$x_{i,e=(v,c)} = l_v + \sum_{e'=(v,c'), c' \neq c} x_{i-1,e'} \tag{1}$$

under the initialization, $x_{0,e'} = 0$ for all edges e' (in the beginning there is no information at the parity check nodes). The summation in (1) is over all edges $e' = (v, c')$ with variable node v except for the target edge $e = (v, c)$. Recall that this is a fundamental property of message passing algorithms [16].

Similarly, for even i and $e = (v, c)$ we have,

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

The final vth output of the network is given by

$$o_v = l_v + \sum_{e'=(v,c')} x_{2L,e'} \tag{3}$$

which is the final marginalization of the BP algorithm.

III. THE PROPOSED DEEP NEURAL NETWORK DECODER

We suggest the following parameterized deep neural network decoder that generalizes the BP decoder of the previous section. We use the same trellis representation for the decoder as in the previous section. The difference is that now we assign weights to the edges in the Tanner graph. These weights will be trained using stochastic gradient descent which is the standard method for training neural networks. More precisely, our decoder has the same trellis architecture as the one defined in the previous section. However, Equations (1), (2) and (3) are replaced by

$$x_{i,e=(v,c)} = \tanh \left(\frac{1}{2} \left(w_{i,v} l_v + \sum_{e'=(v,c'), c' \neq c} w_{i,e,e'} x_{i-1,e'} \right) \right) \tag{4}$$

for odd i,

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

for even i, and

$$o_v = \sigma \left(w_{2L+1,v} l_v + \sum_{e'=(v,c')} w_{2L+1,v,e'} x_{2L,e'} \right) \tag{6}$$

where $\sigma(x) \equiv (1 + e^{-x})^{-1}$ is a sigmoid function. The sigmoid is added so that the final network output is in the range $[0, 1]$. This makes it possible to train the network using a cross entropy loss function, as described in the next section. Apart of the addition of the sigmoid function at the outputs of the network, one can see that by setting all weights to one, Equations (4)-(6) degenerate to (1)-(3). Hence by optimal setting (training) of the parameters of the neural network, its performance can not be inferior to plain BP.

It is easy to verify that the proposed message passing decoding algorithm (4)-(6) satisfies the message passing symmetry conditions [16, Definition 4.81]. Hence, by [16, Lemma 4.90], when transmitting over a binary memoryless symmetric (BMS) channel, the error rate is independent of the transmitted codeword. Therefore, to train the network, it is sufficient to use a database which is constructed by using noisy versions of a single codeword. For convenience we use the zero codeword, which must belong to any linear code. The database reflects various channel output realizations when the zero codeword has been transmitted. The goal is to train the parameters $\{w_{i,v}, w_{i,e,e'}, w_{i,v,e'}\}$ to achieve an N dimensional output word which is as close as possible to the zero codeword. The network architecture is a non-fully connected neural network. We use stochastic gradient descent to train the parameters. The motivation behind the new proposed parameterized decoder is that by setting the weights properly, one can compensate for small cycles in the Tanner graph that represents the code. That is, messages sent by parity check nodes to variable nodes can be weighted, such that if a message is less reliable since it is produced by a parity check node with a large number of small cycles in its local neighborhood, then this message will be attenuated properly.

The time complexity of the deep neural network is similar to plain BP algorithm. Both have the same number of layers and the same number of non-zero weights in the Tanner graph. The deep neural network architecture is illustrated in Figure 1 for a BCH(15,11) code.

IV. EXPERIMENTS

A. Neural Network Training

We built our neural network on top of the TensorFlow framework [17] and used an NVIDIA Tesla K40c GPU for

Feature	Plain BP	Neural BP (Learned BP)
Parameters	None (fixed)	Trainable weights w_i
Output activation	None	Sigmoid (probability)
Learning	No	Yes (via gradient descent)
Connectivity	Sparse (Tanner graph)	Sparse (Tanner graph)
Adaptability	Fixed behavior	Learns reliability of edges
Output	LLR values	Probabilities (0–1)

* In normal BP, all edges are treated equally - no one trusted more or less. So if there are "short cycles" (loops) in the tanner graph, the same information bounce around and confuse the decoder. because BP doesn't know which message is reliable and which is just an echo.

* Reliabe edge → network learns to → larger weight
(message usually correct, high llr) trust it more

* unreliable edge → network learn to → smaller weights
(message from a noisy trust it less
or looping path, low llr)

May goal is address short cycle problem

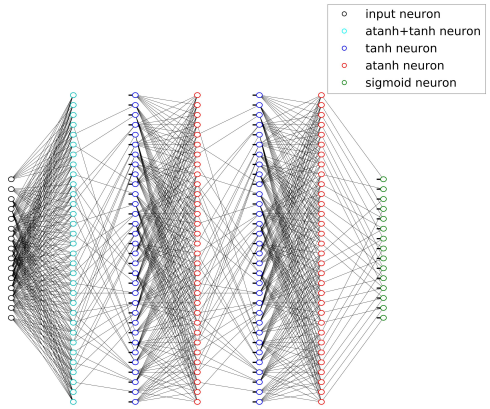


Fig. 1. Deep Neural Network Architecture For BCH(15,11) with 5 hidden layers which correspond to 3 full BP iterations. Note that the self LLR messages l_v are plotted as small bold lines. The first hidden layer and the second hidden layer that were described above are merged together. Also note that this figure shows 3 full iterations and the final marginalization.

accelerated training. We applied cross entropy as our loss function,

$$L(o,y) = -\frac{1}{N}\sum_{v=1}^{N} y_v \log(o_v) + (1-y_v)\log(1-o_v) \quad (7)$$

where o_v, y_v are the deep neural network output and the actual vth component of the transmitted codeword (if the all-zero codeword is transmitted then $y_v = 0$ for all v). Training was conducted using stochastic gradient descent with mini-batches. The mini-batch size was 120 examples. We applied the RMSPROP [18] rule with a learning rate equal to 0.001. The neural network has 10 hidden layers, which correspond to 5 full iterations of the BP algorithm. Each processing element in an odd (even, respectively) indexed hidden layer is described by Equation (4) (Equation (5), respectively). At test time, we inject noisy codewords after transmitting through an AWGN channel and measure the bit error rate (BER) in the decoded codeword at the network output. When computing (4), we also clip the input to the tanh function such that the absolute value of the input is always smaller than some positive constant $A < 10$. This is also required for practical (finite block length) implementations of the BP algorithm, in order to stabilize the operation of the decoder. We trained our decoding network on few different linear codes, including BCH(15,11), BCH(63,36), BCH(63,45) and BCH(127,106).

B. Neural Network Training With Multiloss

(address vanishing gradient problem)

The proposed neural network architecture has the property that after every odd i layer we can add final marginalization. We can use that property to add additional terms in the loss. The additional terms can increase the gradient update at the backpropagation algorithm and allow learning the lower layers. At each odd i layer we add the final marginalization to loss:

$$L(o,y) = -\frac{1}{N}\sum_{i=1,3}^{2L-1}\sum_{v=1}^{N} y_v \log(o_{v,i}) + (1-y_v)\log(1-o_{v,i}) \quad (8)$$

where $o_{v,i}$, y_v are the deep neural network output at the odd i layer and the actual vth component of the transmitted codeword. This network architecture is illustrated in Figure 2.

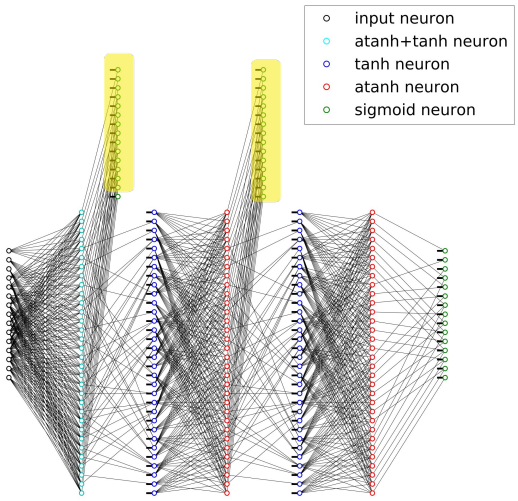


Fig. 2. Deep Neural Network Architecture For BCH(15,11) with training multiloss. Note that the self LLR messages l_v are plotted as small bold lines.

C. Dataset

The training data is created by transmitting the zero codeword through an AWGN channel with varying SNRs ranging from 1dB to 6dB. Each mini batch has 20 codewords for each SNR (a total of 120 examples in the mini batch). For the test data we use codewords with the same SNR range as in the training dataset. Parity check matrices were taken from [19].

D. Results

In this section we present the results of the deep neural decoding networks for various BCH block codes. In each code we observed an improvement compared to the BP algorithm. Note that when we applied our algorithm to the BCH(15,11) code, we obtained close to maximum likelihood results with the deep neural network. For larger BCH codes, the BP algorithm and the deep neural network still have a significant gap from maximum likelihood. The BER figures 3, 4 and 5 show an improvement of up to 0.75dB in the high SNR region. Furthermore, the deep neural network BER is consistently smaller or equal to the BER of the BP algorithm. This result is in agreement with the observation that our network cannot perform worse than the BP algorithm. Figure

6 shows the results of training the deep neural network with multiloss. It shows an improvement of up to 0.9dB compared to the plain BP algorithm. Moreover, we can observe that we can achieve the same BER performance of 50 iteration BP with 5 iteration of the deep neural decoder, This is equal to complexity reduction of factor 10.

We compared the weights of the BP algorithm and the weights of the trained deep neural network for a BCH(63,45) code. We observed that the deep neural network produces weights in the range from -0.8 to 2.2, in contrast to the BP algorithm which has binary 1 or 0 weights. Figure 7 shows the weights histogram for the last layer. Interestingly, the distribution of the weights is close to a normal distribution. In a similar way, every hidden layer in the trained deep neural network has a close to normal distribution. Note that Hinton [20] recommends to initialize the weights with normal distribution. In Figures 8 and 9 we plot the weights of the last hidden layer. Each column in the figure corresponds to a neuron described by Equation (4). It can be observed that most of the weights are zeros except the Tanner graph weights which have value of 1 in Figure 8 (BP algorithm) and some real number in Figure 9 for the neural network. In Figure 8 and 9 we plot a quarter of the weights matrix for better illustration.

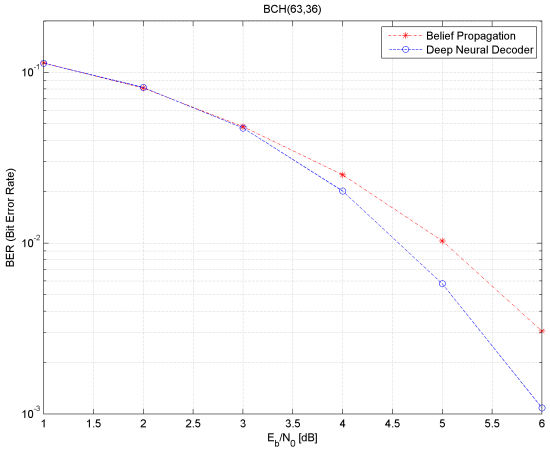


Fig. 3. BER results for BCH(63,36) code.

V. CONCLUSIONS

In this work we applied deep learning techniques to improve the performance of the BP algorithm. We showed that a "soft" Tanner graph can produce improvements when used in the BP algorithm instead of the standard Tanner graph. We believe that the BER improvement was achieved by properly weighting the messages, such that the effect of small cycles in the Tanner graph was partially compensated. It should be emphasized that the parity check matrices that we worked with were obtained from [19]. We have not evaluated our method on parity check matrices for which an attempt was made to reduce the number of small cycles. A notable property of our neural network decoder is that

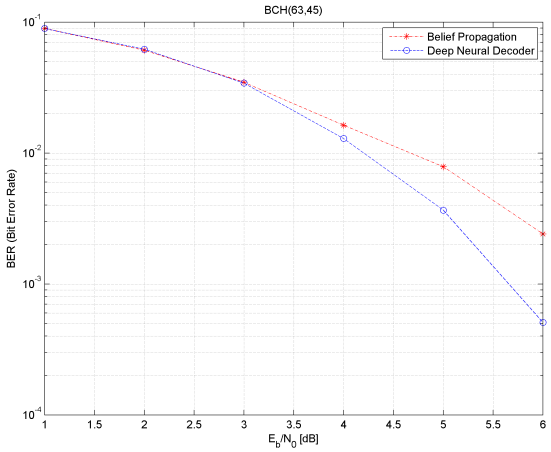


Fig. 4. BER results for BCH(63,45) code.

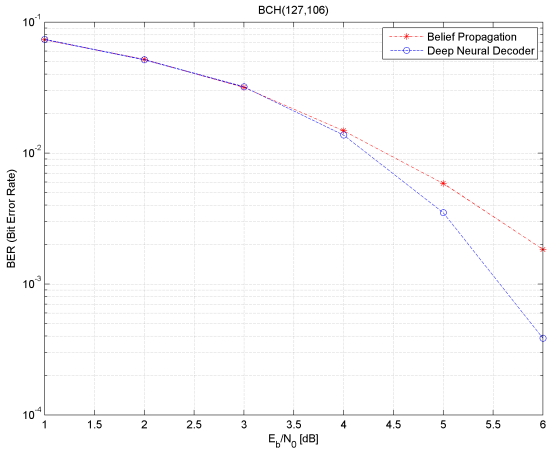


Fig. 5. BER results for BCH(127,106) code.

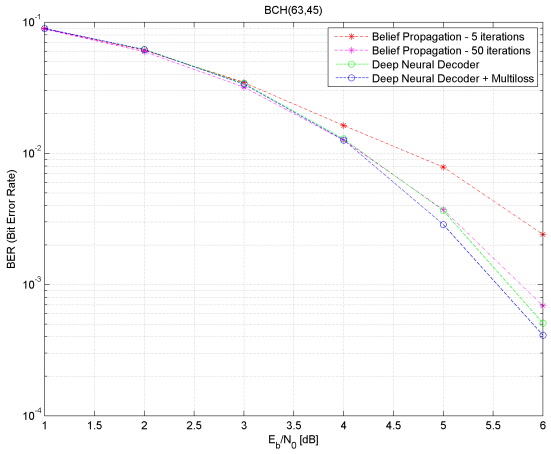


Fig. 6. BER results for BCH(63,45) code trained with multiloss.

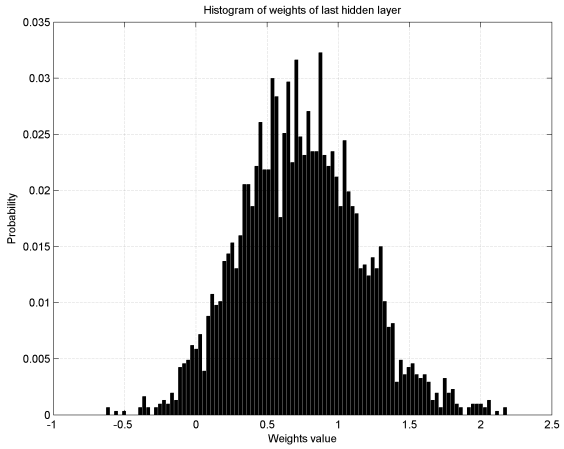


Fig. 7. Weights histogram of last hidden layer of the deep neural network for BCH(63,45) code.

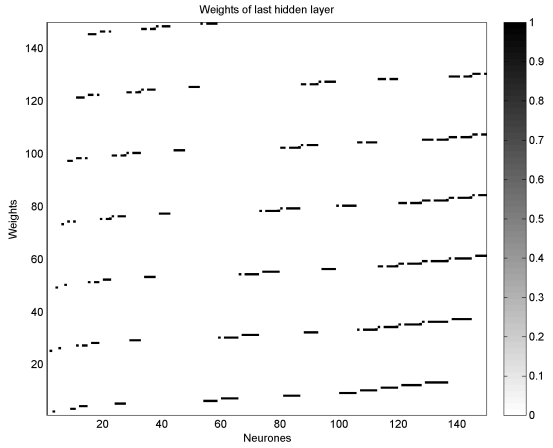


Fig. 8. Weights of the last hidden layer in the BP algorithm for BCH(63,45) code.

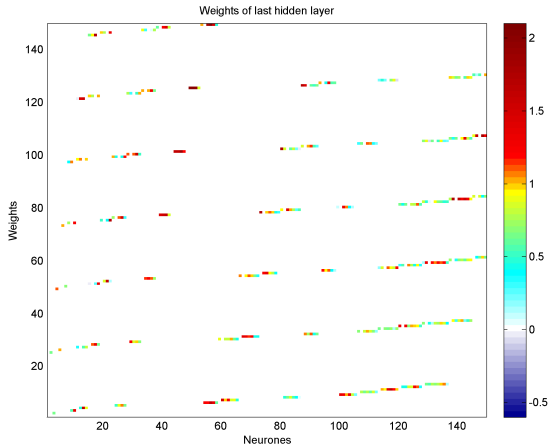


Fig. 9. Weights of the last hidden layer in deep neural network for BCH(63,45) code.

once we have trained its parameters, we can improve performance compared to plain BP without increasing the required computational complexity. Another notable property of the neural network decoder is that we learn the channel and the linear code simultaneously.

We regard this work as a first step in the implementation of deep learning techniques for the design of improved decoders. Our future work include possible improvements in the error rate results by exploring new neural network architectures and combining other decoding methods. Furthermore, we plan to investigate the connection between the parity check matrix and the deep neural network decoding capabilities.

ACKNOWLEDGMENT

This research was supported by the Israel Science Foundation, grant no. 1082/13. The Tesla K40c used for this research was donated by the NVIDIA Corporation.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[2] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[3] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[6] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[8] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[9] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[10] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, Massachusetts: M.I.T. Press, 1963.

[11] J. Jiang and K. R. Narayanan, "Iterative soft-input soft-output decoding of reed-solomon codes by adapting the parity-check matrix," *IEEE Transactions on Information Theory*, vol. 52, no. 8, pp. 3746–3756, 2006.

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

[13] A. Yufit, A. Lifshitz, and Y. Be'ery, "Efficient linear programming decoding of hdpc codes," *IEEE Transactions on Communications*, vol. 59, no. 3, pp. 758–766, 2011.

[14] X. Zhang and P. H. Siegel, "Adaptive cut generation algorithm for improved linear programming decoding of binary linear codes," *IEEE Transactions on Information Theory*, vol. 58, no. 10, pp. 6581–6594, 2012.

[15] M. Helmling, E. Rosnes, S. Ruzika, and S. Scholl, "Efficient maximum-likelihood decoding of linear block codes on binary memoryless channels," in *2014 IEEE International Symposium on Information Theory*. IEEE, 2014, pp. 2589–2593.

[16] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge, UK: Cambridge University Press, 2008.
[17] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems, 2015," *Software available from tensorflow. org*, vol. 1, 2015.
[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] M. Helmling and S. Scholl, "Database of channel codes and ml simulation results," www.uni-kl.de/channel-codes, University of Kaiserslautern, 2016.
[20] G. Hinton, "A practical guide to training restricted boltzmann machines," *Momentum*, vol. 9, no. 1, p. 926, 2010.