ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) | 978-1-7281-7605-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICASSP39728.2021.9414407

# LEARNED DECIMATION FOR NEURAL BELIEF PROPAGATION DECODERS

*(Invited Paper)*

*Andreas Buchberger*[⋆], *Christian Häger*[⋆], *Henry D. Pfister*[†], *Laurent Schmalen*[‡], *and Alexandre Graell i Amat*[⋆]

[⋆] Dept. of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden
[†] Dept. of Electrical and Computer Engineering, Duke University, Durham, North Carolina, USA
[‡] Communications Engineering Lab, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

## ABSTRACT

We introduce a two-stage decimation process to improve the performance of neural belief propagation (NBP), recently introduced by Nachmani *et al.*, for short low-density parity-check (LDPC) codes. In the first stage, we build a list by iterating between a conventional NBP decoder and guessing the least reliable bit. The second stage iterates between a conventional NBP decoder and learned decimation, where we use a neural network to decide the decimation value for each bit. For a (128,64) LDPC code, the proposed NBP with decimation outperforms NBP decoding by 0.75 dB and performs within 1 dB from maximum-likelihood decoding at a block error rate of $10^{-4}$.

## 1. INTRODUCTION

Belief propagation (BP) decoding can be formulated as a sparse deep neural network (NN) where instead of iterating between check nodes (CNs) and variable nodes (VNs), the messages are passed through unrolled iterations in a feed-forward fashion [1, 2]. For short block codes, the weights associated with the edges may counteract the effect of short cycles in the graph by scaling the messages accordingly. This is commonly referred to as neural belief propagation (NBP) and can be seen as a generalization of weighted BP decoding where all individual messages are scaled by a single coefficient [3]. For a given code, the performance of BP and NBP can be improved by using redundant parity-check matrices at the cost of increased complexity [4–10]. A pruning-based NBP decoder was introduced in [10], which improves the performance of NBP at a lower decoding complexity.

It can be observed that the gain of NBP over BP for low-density parity-check (LDPC) codes is mostly in reducing the number of decoding iterations—the performance of NBP and BP converges for a sufficiently large number of iterations and a fundamental gap between (N)BP and maximum-likelihood (ML) decoding remains. Indeed, the error rate of (N)BP decoding of LDPC codes is dominated by absorbing sets from which the (N)BP decoder cannot recover. One way to improve performance is to run multiple decoders with different hard guesses for the least reliable bits [11–14]. Each guess naturally splits the decoding process into two parallel decoders, one for each guess [13, 14]. Similarly, in problems where multiple codewords can have similar posterior probabilities, such as lossy compression, one may make hard decisions for the most reliable bits without splitting the decoder, helping the BP algorithm to converge to a codeword [15–18]. We refer to both of these operations as *decimation* though that term is typically associated with the second case.

In this work, we propose the use of decimation for NBP decoding, thereby introducing a neural belief propagation with decimation (NBP-D) decoder. Our proposed decimation scheme consists of a *list-based decimation* stage and a *learned decimation* stage. We start with the list-based decimation stage and run a conventional NBP decoder for $\ell_{\mathsf{max}}$ iterations. We identify the least reliable VN, i.e., the VN with the lowest absolute *a posteriori* log-likelihood ratio (LLR) and decimate it to $\pm\infty$. Choosing the correct sign is essential as the correct sign will aid convergence whereas the incorrect sign will hinder it. As the correct sign is unknown, we proceed with two graphs—one where the VN is decimated to $+\infty$ and one where the VN is decimated to $-\infty$. We iterate between decimating and decoding using the NBP decoder for the desired number of times and end up with a list of codeword candidates. After the list-based decimation stage is complete, we continue with a learned decimation stage. Similar to the list-based decimation stage, we run a conventional NBP decoder for $\ell_{\mathsf{max}}$ iterations. For each VN, we then use an NN to decide to which value each VN is decimated to. The sign of the VN is not changed. We then iterate between the conventional NBP decoder and the learned decimation for a desired number of decimation steps. We apply our proposed NBP-D decoder to an LDPC code from the CCSDS standard and demonstrate a performance within 1 dB from ML decoding.

## 2. PRELIMINARIES

Consider a linear block code $\mathcal{C}$ of length $n$ and dimension $k$ with parity-check matrix $\boldsymbol{H}$ of size $m \times n$, $m \geq$

This work was partially funded by the Swedish Research Council under grant 2016-04253 and the Knut and Alice Wallenberg Foundation under grant 2018.0090.

$n-k$. The Tanner graph corresponding to the parity-check matrix $\boldsymbol{H}$ is denoted as $\mathcal{G} = (\mathcal{V}_\mathsf{v}, \mathcal{V}_\mathsf{c}, \mathcal{E})$, consisting of a set $\mathcal{V}_\mathsf{c} = \{\mathsf{c}_1, \ldots, \mathsf{c}_m\}$ of $m$ CNs, a set $\mathcal{V}_\mathsf{v} = \{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$ of $n$ VNs, and a set of edges $\mathcal{E}$ connecting CNs with VNs. For each VN $\mathsf{v} \in \mathcal{V}_\mathsf{v}$ we define its neighborhood $\mathcal{N}(\mathsf{v}) \triangleq \{\mathsf{c} \in \mathcal{V}_\mathsf{c} : (\mathsf{v},\mathsf{c}) \in \mathcal{E}\}$, i.e., the set of all CNs connected to VN $\mathsf{v}$, and equivalently, the neighborhood of a CN $\mathsf{c} \in \mathcal{V}_\mathsf{c}$ is defined as $\mathcal{N}(\mathsf{c}) \triangleq \{\mathsf{v} \in \mathcal{V}_\mathsf{v} : (\mathsf{v},\mathsf{c}) \in \mathcal{E}\}$. Let $\mu^{(\ell)}_{\mathsf{v}_i \to \mathsf{c}_j}$ be the message passed from VN $\mathsf{v}_i \in \mathcal{V}_\mathsf{v}$ to CN $\mathsf{c}_j \in \mathcal{V}_\mathsf{c}$ and $\mu^{(\ell)}_{\mathsf{c}_j \to \mathsf{v}_i}$ the message passed from CN $\mathsf{c}_j \in \mathcal{V}_\mathsf{c}$ to VN $\mathsf{v}_i \in \mathcal{V}_\mathsf{v}$ in the $\ell$-th decoding iteration. For BP decoding, the VN and CN updates are

$$\mu^{(\ell)}_{\mathsf{v}_i \to \mathsf{c}_j} = \mu_{\mathsf{ch},\mathsf{v}_i} + \sum_{\mathsf{c} \in \mathcal{N}(\mathsf{v}_i) \setminus \mathsf{c}_j} \mu^{(\ell)}_{\mathsf{c} \to \mathsf{v}_i} \tag{1}$$

and

$$\mu^{(\ell)}_{\mathsf{c}_j \to \mathsf{v}_i} = 2 \tanh^{-1} \left( \prod_{\mathsf{v} \in \mathcal{N}(\mathsf{c}_j) \setminus \mathsf{v}_i} \tanh\left( \frac{1}{2} \mu^{(\ell)}_{\mathsf{v} \to \mathsf{c}_j} \right) \right) \tag{2}$$

respectively, where $\mu_{\mathsf{ch},\mathsf{v}_i}$ is the channel message. For binary transmission over the additive white Gaussian noise channel

$$\mu_{\mathsf{ch},\mathsf{v}_i} \triangleq \ln\left( \frac{p_{Y|B}(y_i|b_i=0)}{p_{Y|B}(y_i|b_i=1)} \right) = \frac{2y_i}{\sigma^2}$$

where $y_i$ is the channel output, $b_i$ is the transmitted bit, and $\sigma^2$ is the noise variance. The *a posteriori* LLR in the $\ell$-th iteration is

$$\mu^{(\ell)}_{\mathsf{v}_i} = \mu_{\mathsf{ch},\mathsf{v}_i} + \sum_{\mathsf{c} \in \mathcal{N}(\mathsf{v}_i)} \mu^{(\ell)}_{\mathsf{c} \to \mathsf{v}_i}.$$

### 2.1. Neural Belief Propagation

For conventional BP, the decoder iterates between VNs and CNs by passing messages along the connecting edges. For a given maximum number of iterations $\ell_{\mathsf{max}}$, one can *unroll* the graph by stacking $\ell_{\mathsf{max}}$ copies of the Tanner graph, and perform message passing over the unrolled graph. One way to counteract the effect of short cycles on the performance of BP decoding for short linear block codes is to introduce weights for each edge of the unrolled Tanner graph [1,2]. The resulting unrolled weighted graph can be interpreted as a *sparse* NN and accordingly decoding over the unrolled graph is referred to as NBP. For NBP, the update rules (1) and (2) are modified to

$$\mu^{(\ell)}_{\mathsf{v}_i \to \mathsf{c}_j} = w^{(\ell)}_{\mathsf{v}_i \to \mathsf{c}_j} \left( w^{(\ell)}_{\mathsf{ch},\mathsf{v}_i} \mu_{\mathsf{ch},\mathsf{v}_i} + \sum_{\mathsf{c} \in \mathcal{N}(\mathsf{v}_i) \setminus \mathsf{c}_j} \mu^{(\ell)}_{\mathsf{c} \to \mathsf{v}_i} \right) \tag{3}$$

and

$$\mu^{(\ell)}_{\mathsf{c}_j \to \mathsf{v}_i} = 2 w^{(\ell)}_{\mathsf{c}_j \to \mathsf{v}_i} \tanh^{-1} \left( \prod_{\mathsf{v} \in \mathcal{N}(\mathsf{c}_j) \setminus \mathsf{v}_i} \tanh\left( \frac{1}{2} \mu^{(\ell)}_{\mathsf{v} \to \mathsf{c}_j} \right) \right) \tag{4}$$

where $w^{(\ell)}_{\mathsf{ch},\mathsf{v}}$, $w^{(\ell)}_{\mathsf{v} \to \mathsf{c}}$, and $w^{(\ell)}_{\mathsf{c} \to \mathsf{v}}$, are the channel weights, the weights on the edges connecting VNs to CNs, and the weights on the edges connecting CNs to VNs, respectively. The *a posteriori* LLR in the $\ell$-th iteration is

$$\mu^{(\ell)}_{\mathsf{v}_i} = w^{(\ell)}_{\mathsf{ch},\mathsf{v}_i} \mu_{\mathsf{ch},\mathsf{v}_i} + \sum_{\mathsf{c} \in \mathcal{N}(\mathsf{v}_i)} \mu^{(\ell)}_{\mathsf{c} \to \mathsf{v}_i}.$$

In (3) and (4) the weights are untied over all nodes as well as over all iterations, i.e., each edge has an individual weight. In order to reduce complexity and storage requirements for NBP, the weights can also be tied. In [2,19], tying the weights temporally, i.e., over iterations, and spatially, i.e., all edges within a layer have the same weight, was explored.

For finding the edge weights, one may consider a binary classification task for each of the $n$ bits. As a loss function, the average bitwise cross-entropy between the transmitted bits and the output LLRs of the final VN layer can be used [1,2]. The optimization behavior can be improved by using a multi-loss function [1,2], where the overall loss is the average bitwise cross-entropy between the transmitted bits and the output LLRs of each VN layer. Using the loss function, stochastic gradient descent (and variants thereof) can be used to optimize the weights.

## 3. DECIMATED NEURAL BELIEF PROPAGATION DECODER

While NBP improves upon BP decoding for a fixed (relatively small) number of iterations, for LDPC codes NBP and BP appear to yield the same performance for large enough number of iterations, and a gap to the ML performance remains. To overcome this limitation of NBP, here we propose a two-stage decimation process on top of NBP decoding consisting of a *list-based decimation* stage and a *learned decimation* stage. In both stages, we iterate between a decimation process and a conventional NBP decoder with $\ell_{\mathsf{max}}$ iterations for which we consider the case where the weights are tied over iterations, i.e, $w^{(\ell)}_{\mathsf{ch},\mathsf{v}_i} = w_{\mathsf{ch},\mathsf{v}_i}$, $w^{(\ell)}_{\mathsf{v}_i \to \mathsf{c}_j} = w_{\mathsf{v}_i \to \mathsf{c}_j}$, and $w^{(\ell)}_{\mathsf{c}_j \to \mathsf{v}_i} = w_{\mathsf{c}_j \to \mathsf{v}_i}$, and are optimized for a large number of iterations. We denote our proposed decoder as NBP-D($\ell_{\mathsf{max}}, n_{\mathsf{D}}, n_{\mathsf{LD}}$), where $\ell_{\mathsf{max}}$ denotes the number of iterations of the conventional NBP decoder, $n_{\mathsf{D}}$ is the number of decimations in the list-based decimation stage, and $n_{\mathsf{LD}}$ denotes the number of decimations in the learned decimation stage. In the following, we describe the two decimation stages in detail, provide the training procedure, and give a brief complexity discussion.

### 3.1. List-based Decimation Stage

We start by running NBP for $\ell_{\mathsf{max}}$ iterations. We then identify the VN $\mathsf{v}$ with the lowest absolute *a posteriori* LLR $\mu^{(\ell_{\mathsf{max}})}_{\mathsf{v}}$, i.e., the least reliable LLR and decimate it to $\pm\infty$, i.e., set $\mu_{\mathsf{ch},\mathsf{v}} = \pm\infty$. Since the correct sign is unknown, we build a

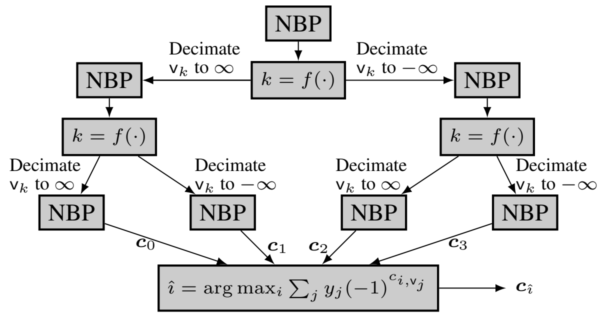


**Fig. 1**. Decoding tree for $n_{\mathsf{D}} = 2$ and $n_{\mathsf{LD}} = 0$ where $f(\cdot) = \arg\min_j |\mu_{\mathsf{v},j}^{(\ell_{\max})}|$.

decoding tree: Each time we decimate a VN, we create two new graphs where in one the VN is decimated to $+\infty$ and in the other the VN is decimated to $-\infty$. We then run the NBP decoder for each of the two graphs. We alternate between running NBP and decimation $n_{\mathsf{D}}$ times. In Fig. 1, we depict the decoding tree for $n_{\mathsf{D}} = 2$.

### 3.2. Learned Decimation Stage

While using the previously described list-based decimation allows to significantly boost the performance, it has the drawback that the complexity increases exponentially in the number of decimations. To reduce the size of the decoding tree, we propose an NN-based approach. More precisely, instead of continuing to unfold the decoding tree, we use an NN to decide to which value each VN should be decimated. The NN takes the incoming messages to the node and the channel LLR as input and outputs a value whose absolute value is then added to the channel message. The sign is kept according to the output LLR of the respective node. Hence, we decimate each VN $\mathsf{v} \in \mathcal{V}_{\mathsf{v}}$ according to

$$\mu_{\mathsf{ch},\mathsf{v}} = \mu_{\mathsf{ch},\mathsf{v}} + \operatorname{sign}\left(\mu_{\mathsf{v}}^{(\ell_{\max})}\right) \cdot \left| f_{\mathsf{NN}}\left(\mu_{\mathsf{ch},\mathsf{v}}, \{\mu_{\mathsf{c}\to\mathsf{v}}^{(\ell_{\max})} | \mathsf{c} \in \mathcal{N}(\mathsf{v})\}, \boldsymbol{\theta}\right)\right|,$$

where $f_{\mathsf{NN}}(\cdot)$ denotes the NN and $\boldsymbol{\theta}$ denotes all trainable parameters of the NN. The same NN is applied for all VNs and weights are shared between VNs and all decimation steps. We alternate learned decimation and running NBP $n_{\mathsf{LD}}$ times.

After the $n_{\mathsf{D}}$ list-based decimations and $n_{\mathsf{LD}}$ learned decimations, we obtain $2^{n_{\mathsf{D}}}$ codeword candidates $\boldsymbol{c}_i$ and we choose the most likely codeword. The NBP-D decoder is described in Algorithm 1.

### 3.3. Training

We perform training in two stages. We first consider a conventional NBP decoder with weights tied over the iterations. We train this decoder using the multiloss bitwise cross-entropy between the output LLRs of each VN layer and the correct codeword as the loss function and the Adam optimizer [2]. We then freeze the learned weights and unroll the NBP-D($\ell_{\max}, n_{\mathsf{D}}, n_{\mathsf{LD}}$) decoder. To reduce the complexity of the training procedure, we only consider the correct path through the decoding tree in the list-based decimation stage, i.e., we assume a genie that provides the correct sign of the decimated VN. This incurs no loss in block error rate (BLER) as, of the $2^{n_{\mathsf{D}}}$ decoders, only the one with the correct decisions can return the correct codeword. We train the unrolled NBP-D decoder using the multiloss bitwise cross-entropy between the output LLRs of each VN layer and the correct codeword as the loss function and the Adam optimizer.

**Algorithm 1** Neural belief propagation with decimation.

**Input:** Trained NBP decoder with $\ell_{\max}$ iterations
Channel outputs $y_i$ and $\mu_{\mathsf{ch},\mathsf{v}_i}$ for all VNs $\mathsf{v}_i \in \mathcal{V}_{\mathsf{v}}$
# of decimations in the list-based decimation stage $n_{\mathsf{D}}$
# of decimations in the learned decimation stage $n_{\mathsf{LD}}$

$\mathcal{M}_1 \leftarrow \{\mu_{\mathsf{ch},\mathsf{v}}\}$ and $\mathcal{M}_i \leftarrow \{\} \, \forall\, i \in \{2, \ldots, 2^{n_{\mathsf{D}}}\}$

▷ *List-based Decimation Stage (Section 3.1)*
**for** $i \leftarrow \{1, \ldots, n_{\mathsf{D}}\}$ **:**
  **for** $j \leftarrow \{1, \ldots, 2^{i-1}\}$ **:**
    Decode $\mathcal{M}_j$ using NBP to $\mu_{\mathsf{v}}^{(\ell_{\max})}$
    $k \leftarrow \arg\min_i |\mu_{\mathsf{v}_i}^{(\ell_{\max})}|$
    $\mathcal{M}_{2^{i-1}+j} \leftarrow \mathcal{M}_j$
    $\mu_{\mathsf{ch},\mathsf{v}_k} \leftarrow +\infty$ with $\mu_{\mathsf{ch},\mathsf{v}_k} \in \mathcal{M}_j$
    $\mu_{\mathsf{ch},\mathsf{v}_k} \leftarrow -\infty$ with $\mu_{\mathsf{ch},\mathsf{v}_k} \in \mathcal{M}_{2^{i-1}+j}$
  **end for**
**end for**

▷ *Learned Decimation Stage (Section 3.2)*
**for** $i \leftarrow \{1, \ldots, n_{\mathsf{LD}}\}$ **:**
  **for** $j \leftarrow \{1, \ldots, 2^{n_{\mathsf{D}}}\}$ **:**
    Decode $\mathcal{M}_j$ using NBP to $\mu_{\mathsf{v}}^{(\ell_{\max})}$
    **for** $\mathsf{v} \in \mathcal{V}_{\mathsf{v}}$ **:**
      $\mu_{\mathsf{ch},\mathsf{v}} \leftarrow \mu_{\mathsf{ch},\mathsf{v}} + \operatorname{sign}(\mu_{\mathsf{v}}) \left| f_{\mathsf{NN}}\left(\mu_{\mathsf{ch},\mathsf{v}}, \{\mu_{\mathsf{c}\to\mathsf{v}}^{(\ell_{\max})}, \mathsf{c} \in \mathcal{N}(\mathsf{v})\}, \boldsymbol{\theta}\right)\right|$
    **end for**
  **end for**
**end for**

**for** $j \leftarrow \{1, \ldots, 2^{n_{\mathsf{D}}}\}$ **:**
  Decode $\mathcal{M}_j$ using NBP to $\mu_{\mathsf{v}}^{(\ell_{\max})}$
  $\boldsymbol{c}_j \leftarrow (c_{j,\mathsf{v}_1}, \ldots, c_{j,\mathsf{v}_n})$, $c_{j,\mathsf{v}} = \left(1 - \operatorname{sign}\left(\mu_{\mathsf{v}}^{(\ell_{\max})}\right)\right)/2$
**end for**
$\hat{\imath} = \arg\max_i \sum_j y_j (-1)^{c_{i,\mathsf{v}_j}}$
**return** $\boldsymbol{c}_{\hat{\imath}}$

### 3.4. Complexity Discussion

On a high level, by disregarding hardware implementation details, the CN update is the most complex operation due to the evaluation of the $\tanh$ and inverse $\tanh$ functions. A commonly used complexity measure is given by $\bar{d}_{\mathsf{c}} m \ell_{\max}$ [20], where $\bar{d}_{\mathsf{c}}$ is the average CN degree and $m$ the number of CNs.

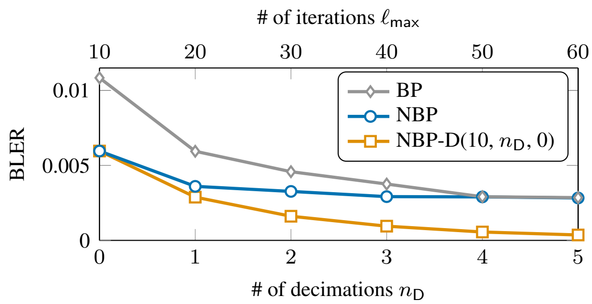


**Fig. 2**. BLER at $E_b/N_0 = 4\,$dB.

The complexity of the NBP-D decoder follows as

$$\bar{d}_c m \ell_{\max} \left(2^{n_D+1} - 1 + n_{LD} 2^{n_D}\right).$$

The memory requirement for the decoders is dominated by the number of weights that need to be stored. For the NBP decoder, this corresponds to the number of edges in the Tanner graph. In the case of NBP-D, additionally the weights of the NN need to be taken into account. Note that the weights also entail additional multiplications.

## 4. NUMERICAL RESULTS

In this work, we consider an LDPC code of length $n = 128$ and rate $0.5$ with average CN degree $\bar{d}_c = 8$ as defined by the CCSDS standard. Its parity-check matrix is of size $64 \times 128$. It is important to note that the presented concepts are not limited to a specific code and extend to any other sparse code.

For training the NBP and NBP-D decoders, the batch size was set to $128$, the learning rate to $0.001$, and the Adam optimizer was used for gradient updates. The NN of the NBP-D decoder consists of three layers where the first and second layer contain $16$ neurons and use the ReLU activation function, and the third layer consists of a single neuron with no activation function (i.e., a linear layer).

In Fig. 2, we plot the BLER as a function of the number of iterations for conventional BP, NBP, and NBP-D for which $n_{LD} = 0$, i.e., no learned decimation is employed. For a fixed number of iterations, NBP outperforms BP. For LDPC codes, the gain of NBP over BP appears to vanish with an increasing number of iterations, and at $50$ iterations BP and NBP have virtually the same performance. Importantly, increasing the number of iterations even further does not result in an improved performance. Considering the NBP-D decoder, the decimation enables us to outperform (N)BP, even for a single decimation. It is important to note that while the complexity of (N)BP increases linearly in the number of iterations, the complexity of NBP-D increases exponentially in the number of decimations $n_D$.

In Fig. 3, we show the BLER as a function of $E_b/N_0$. NBP with $50$ iterations (NBP$(50)$) performs about $1.75\,$dB from ML. Increasing the number of iterations even further would not result in a better performance (see Fig. 2). NBP-D$(10, 4, 0)$ improves by $0.4\,$dB over NBP$(50)$ and matches the performance of pruning-based NBP $\mathcal{D}_3$ in [10]. Adding a single learned decimation (NBP-D$(10, 4, 1)$) improves the performance by $0.3\,$dB. Allowing four learned decimation steps (NBP-D$(10, 4, 4)$), the gap to ML is further reduced to slightly less than $1\,$dB. The choice of $n_D = 4$ is a trade-off between performance and complexity. Increasing $n_D \to n$ will eventually lead to ML decoding. Relying solely on learned decimation ($n_D = 0$) is not competitive as the correct choice of the sign for VNs of low reliability is never guaranteed. In Table 1, we compare the complexities for the decoders in Fig. 3. The numbers in parentheses indicate how much more complex a decoder is with reference to NBP$(50)$.

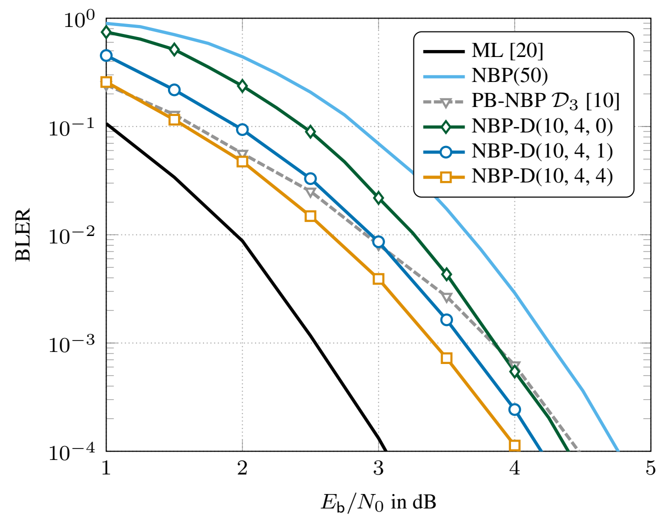


**Fig. 3**. BLER for the CCSDS $(128, 64)$ LDPC code.

**Table 1**. Complexity of the decoders in Fig. 3. The value in parentheses indicates how many times more complex the decoder is compared to NBP with 50 iterations.

| | Complexity | | # of weights |
|---|---|---|---|
| NBP(50) | 25600 | (1.0) | 1152 |
| PB-NBP $\mathcal{D}_3$ [10] | 25920 | (1.0) | 28416 |
| NBP-D$(10, 4, 0)$ | 158720 | (6.2) | 1152 |
| NBP-D$(10, 4, 1)$ | 240640 | (9.4) | 1553 |
| NBP-D$(10, 4, 4)$ | 486400 | (19) | 1553 |

## 5. CONCLUSION

We proposed a neural belief propagation with decimation (NBP-D) decoder for LDPC codes. By combining a list-based decimation stage and a learned decimation stage where a neural network learns which VNs to decimate, we have shown that we can significantly improve the performance of BP and neural belief propagation (NBP), and achieve a performance within $1\,$dB from ML. The concept of NBP-D can be applied to any other short block code. For sparse codes, similar performance improvements can be expected.

## 6. REFERENCES


[1] Eliya Nachmani, Yair Be'ery, and David Burshtein, "Learning to decode linear codes using deep learning," in *Proc. Annu. Allerton Conf. Commun., Control, Comput.*, Allerton, IL, USA, Sep. 2016, pp. 341–346.

[2] Eliya Nachmani, Elad Marciano, Loren Lugosch, Warren J. Gross, David Burshtein, and Yair Be'ery, "Deep learning methods for improved decoding of linear codes," *IEEE J. Sel. Topics Signal Process.*, vol. 12, no. 1, pp. 119–131, Feb. 2018.

[3] Jinghu Chen and Marc P. C. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Trans. Commun.*, vol. 50, no. 3, pp. 406–414, Mar. 2002.

[4] Martin Bossert and Ferdinand Hergert, "Hard- and soft-decision decoding beyond the half minimum distance—an algorithm for linear codes," *IEEE Trans. Inf. Theory*, vol. 32, no. 5, pp. 709–714, Sep. 1986.

[5] Aditi Kothiyal, Oscar Y. Takeshita, Wenyi Jin, and Marc Fossorier, "Iterative reliability-based decoding of linear block codes with adaptive belief propagation," *IEEE Commun. Lett.*, vol. 9, no. 12, pp. 1067–1069, Dec. 2005.

[6] Jing Jiang and Krisha R. Narayanan, "Iterative soft-input soft-output decoding of Reed-Solomon codes by adapting the parity-check matrix," *IEEE Trans. Inf. Theory*, vol. 52, no. 8, pp. 3746–3756, Aug. 2006.

[7] Thomas R. Halford and Keith M. Chugg, "Random redundant soft-in soft-out decoding of linear block codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Seattle, WA, USA, Jun. 2006, pp. 2230–2234.

[8] Thorsten Hehn, Johannes Huber, Olgica Milenkovic, and Stefan Laendner, "Multiple-bases belief-propagation decoding of high-density cyclic codes," *IEEE Trans. Commun.*, vol. 58, no. 1, pp. 1–8, Jan. 2010.

[9] Elia Santi, Christian Häger, and Henry D. Pfister, "Decoding Reed-Muller codes using minimum-weight parity checks," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Vail, CO, USA, Jun. 2018, pp. 1296–1300.

[10] Andreas Buchberger, Christian Häger, Henry Pfister, Laurent Schmalen, and Alexandre Graell i Amat, "Pruning neural belief propagation decoders," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Los Angeles, CA, USA, Jun. 2020, pp. 338–342.

[11] David Chase, "A class of algorithms for decoding block codes with channel measurement information," *IEEE Trans. Inf. Theory*, vol. 18, no. 1, pp. 170–182, Jan. 1972.

[12] Hossein Pishro-Nik and Faramarz Fekri, "On decoding of low-density parity-check codes over the binary erasure channel," *IEEE Trans. Inf. Theory*, vol. 50, no. 3, pp. 439–454, Mar. 2004.

[13] Nedeljko Varnica, Marc P. C. Fossorier, and Aleksandar Kav̌cić, "Augmented belief propagation decoding of low-density parity check codes," *IEEE Trans. Commun.*, vol. 55, no. 7, pp. 1308–1317, Jul. 2007.

[14] Cyril Méasson, Andrea Montanari, and Rüdiger Urbanke, "Maxwell construction: The hidden bridge between iterative and maximum a posteriori decoding," *IEEE Trans. Inf. Theory*, vol. 54, no. 12, pp. 5277–5307, Dec. 2008.

[15] Tomáš Filler and Jessica Fridrich, "Binary quantization using belief propagation with decimation over factor graphs of LDGM codes," in *Proc. Annu. Allerton Conf. Commun., Control, Comput.*, Monticello, IL, USA, Sep. 2007.

[16] Martin J. Wainwright, Elitza Maneva, and Emin Martinian, "Lossy source compression using low-density generator matrix codes: Analysis and algorithms," *IEEE Trans. Inf. Theory*, vol. 56, no. 3, pp. 1351–1368, Mar. 2010.

[17] Daniel Castanheira and Atilio Gameiro, "Lossy source coding using belief propagation and soft-decimation over LDGM codes," in *Proc.Int. Symp. Pers. Indoor Mobile Radio Commun. (PIMRC)*, Istanbul, Turkey, Sep. 2010, pp. 413–436.

[18] Vahid Aref, Nicolas Macris, and Marc Vuffray, "Approaching the rate-distortion limit with spatial coupling, belief propagation, and decimation," *IEEE Trans. Inf. Theory*, vol. 61, no. 7, pp. 3954–3979, Jul. 2015.

[19] Mengke Lian, Fabrizio Carpi, Christian Häger, and Henry D. Pfister, "Learned belief-propagation decoding with simple scaling and SNR adaptation," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Paris, France, Jul. 2019, pp. 161–165.

[20] Benjamin Smith, Masoud Ardakani, Wei Yu, and Frank R. Kschischang, "Design of irregular LDPC codes with optimized performance-complexity tradeoff," *IEEE Trans. Commun.*, vol. 58, no. 2, pp. 489–499, Feb. 2010.

[21] Michael Helmling, Stefan Scholl, Florian Gensheimer, Tobias Dietz, Kira Kraft, Stefan Ruzika, and Norbert Wehn, "Database of channel codes and ML simulation results," www.uni-kl.de/channel-codes, 2019.