# Learned Belief-Propagation Decoding with Simple Scaling and SNR Adaptation

Mengke Lian*, Fabrizio Carpi†, Christian Häger*‡, and Henry D. Pfister*
*Department of Electrical and Computer Engineering, Duke University, Durham, North Carolina
†Department of Engineering and Architecture, University of Parma, Parma, Italy
‡Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden

***Abstract*—We consider the weighted belief-propagation (WBP) decoder recently proposed by Nachmani et al. where different weights are introduced for each Tanner graph edge and optimized using machine learning techniques. Our focus is on simple-scaling models that use the same weights across certain edges to reduce the storage and computational burden. The main contribution is to show that simple scaling with few parameters often achieves the same gain as the full parameterization. Moreover, several training improvements for WBP are proposed. For example, it is shown that minimizing average binary cross-entropy is suboptimal in general in terms of bit error rate (BER) and a new "soft-BER" loss is proposed which can lead to better performance. We also investigate parameter adapter networks (PANs) that learn the relation between the signal-to-noise ratio and the WBP parameters. As an example, for the $(32, 16)$ Reed–Muller code with a highly redundant parity-check matrix, training a PAN with soft-BER loss gives near-maximum-likelihood performance assuming simple scaling with only three parameters.**

## I. Introduction

Recent progress in machine learning and off-the-shelf learning packages have made it tractable to add many parameters to existing communication algorithms and optimize. One example of this approach is the weighted belief-propagation (WBP) decoder recently proposed by Nachmani, Be'ery, and Burshtein [1], where different weights (or scale factors) are introduced for each edge in the Tanner graph. These weights are then optimized empirically using tools and software from deep learning. Their results show that WBP can provide significant gains over standard BP when applied to the parity-check (PC) matrices of short BCH codes. A more comprehensive treatment of this idea can be found in [2].

While the performance gains of WBP decoding are worth investigating, the additional complexity of optimizing, storing, and applying one weight per edge is significant. In this paper, we focus on *simple-scaling* models for WBP that share weights across edges to reduce the storage and computational burden. In these models, only three scalar parameters are used per iteration: a message weight, a channel weight, and a damping factor. We show that such simple-scaling models are often sufficient to obtain gains similar to the full parameterization.

The work of M. Lian and H. D. Pfister was supported in part by the National Science Foundation (NSF) under Grant No. 1718494. The work of C. Häger was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant No. 749798. Any opinions, findings, conclusions, and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these sponsors. Please send correspondence to henry.pfister@duke.edu.

For WBP, the average binary cross-entropy across bit positions is used as the optimization loss function in [1], [2]. This approach is also adopted in related works, see, e.g., [3], [4]. On the other hand, we show that minimizing average binary cross-entropy does not necessarily minimize the bit error rate (BER). We propose a new loss function which can be regarded as a "soft" version of BER. This loss function can lead to performance gains when optimizing WBP with highly redundant PC matrices, e.g., for Reed–Muller codes.

As a last contribution, we propose a simple solution to the problem that the optimal WBP parameters may be different for different signal-to-noise ratios (SNRs). In particular, we use a parameter adapter network (PAN) that learns the relation between the SNR and the optimal WBP parameters. The usefulness of this approach is illustrated with several examples.

## II. Background

Consider an $(N, K)$ binary linear code $\mathcal{C}$ defined by an $M \times N$ PC matrix $\mathbf{H}$, where $N$ is the code length, $K$ is the code dimension, and $M \geq N - K$. We assume transmission over the additive white Gaussian noise channel according to $y_v = (-1)^{x_v} + z_v$, where $y_v$ is the $v$-th output symbol, $x_v$ is the corresponding bit in the transmitted codeword $\boldsymbol{x}$, $z_v \sim \mathcal{N}(0, (2RE_\mathrm{b}/N_0)^{-1})$, and $R = K/N$ is the code rate. We refer to $\rho \triangleq E_\mathrm{b}/N_0$ as the signal-to-noise ratio (SNR).

Given any PC matrix $\mathbf{H}$, one can construct a bipartite Tanner graph $\mathcal{G} = (V, C, E)$, where $V = \{1, 2, \ldots, N\} \triangleq [N]$ and $C = [M]$ are sets of variable nodes and check nodes. The edges, $E = \{(v, c) \in V \times C \,|\, H_{cv} \neq 0\}$, connect all parity checks to the variables involved in them. By convention, the boundary symbol $\partial$ denotes the neighborhood operator defined by $\partial v \triangleq \{c \,|\, (v, c) \in E\}$ and $\partial c \triangleq \{v \,|\, (v, c) \in E\}$.

### A. Weighted Belief-Propagation Decoding

WBP is an iterative algorithm that passes messages in the form of log-likelihood ratios (LLRs) along the edges of $\mathcal{G}$ [1]. In the variable-to-check-node step, the pre-update message is

$$\lambda_{v\to c}^{\prime(t)} = w_v^{(t)} \ell_v + \sum_{c' \in \partial v \setminus c} w_{vc'}^{(t)} \hat{\lambda}_{c'\to v}^{(t-1)}, \tag{1}$$

where $w_{vc}^{(t)}$ is a weight assigned to the edge $(v, c)$ and $w_v^{(t)}$ is a weight assigned to the channel message

$$\ell_v \triangleq \log\left(\frac{\Pr(y_v \,|\, x_v = 0)}{\Pr(y_v \,|\, x_v = 1)}\right) = 4R\rho y_v. \tag{2}$$

In the check-to-variable-node step, the pre-update message is

$$\hat{\lambda}'^{(t)}_{c\to v} = 2\tanh^{-1}\left(\prod_{v'\in\partial c\setminus v}\tanh\left(\frac{\lambda^{(t)}_{v'\to c}}{2}\right)\right). \tag{3}$$

To mitigate oscillations and enhance convergence, we apply *damping* to complete the message updates [5]. In particular, the final messages are a convex combination of the previous value and the pre-update value according to

$$\lambda^{(t)}_{v\to c} = \gamma\lambda^{(t-1)}_{v\to c} + (1-\gamma)\lambda'^{(t)}_{v\to c}, \tag{4}$$
$$\hat{\lambda}^{(t)}_{c\to v} = \gamma\hat{\lambda}^{(t-1)}_{c\to v} + (1-\gamma)\hat{\lambda}'^{(t)}_{c\to v}, \tag{5}$$

where $\gamma\in[0,1]$ is the damping factor and $\lambda^{(0)}_{v\to c} = \hat{\lambda}^{(0)}_{c\to v} = 0$ for all $(v,c)\in E$.[1] Finally, output LLRs are computed as

$$m^{(t)}_v = w^{(t)}_v\ell_v + \sum_{c'\in\partial v} w^{(t)}_{vc'}\hat{\lambda}^{(t)}_{c'\to v}. \tag{6}$$

The sigmoid function $\sigma(x) = (1+e^{-x})^{-1}$ maps $m^{(t)}_v$ to an estimate of the probability that $x_v = 1$ according to $o^{(t)}_v = 1-\sigma(m^{(t)}_v)$. The corresponding hard decision is denoted by $\hat{o}^{(t)}_v$. For convenience, we also define $o_v \triangleq o^{(T)}_v$, $\hat{o}_v \triangleq \hat{o}^{(T)}_v$, and $m_v \triangleq m^{(T)}_v$, where $T$ is the total number of iterations. Setting all weights to 1 and $\gamma = 0$ recovers standard BP.

### B. Random Redundant Decoding

The performance of BP can be improved by using redundant PC matrices where $M > N-K$. This can for example be realized by adding dual codewords as rows to a standard PC matrix [6]. Another approach, referred to as random redundant decoding (RRD), is to use different PC matrices in each iteration [7], [8]. This can be implemented efficiently by exploiting the code's automorphism group. In particular, let $\mathcal{S}_N$ be the symmetric group on $N$ elements, i.e., $\pi\in\mathcal{S}_N$ is a permutation on $[N]$. The automorphism group of a code $\mathcal{C}$ is defined as $\mathsf{Aut}(\mathcal{C}) \triangleq \{\pi\in\mathcal{S}_N \,|\, \boldsymbol{x}^\pi\in\mathcal{C}, \forall\,\boldsymbol{x}\in\mathcal{C}\}$, where $\boldsymbol{x}^\pi$ denotes a permuted vector, i.e., $x^\pi_i = x_{\pi(i)}$. RRD cascades $T_{\text{out}}$ BP decoders, each run with $T_{\text{in}}$ iterations, where the permuted output LLRs of the last decoder serve as soft input for the next decoder. A new, randomly sampled permutation $\pi_\tau\in\mathsf{Aut}(\mathcal{C})$ is used for each outer iteration $\tau\in[T_{\text{out}}]$. This effectively uses $T_{\text{out}}$ different PC matrices while fixing the Tanner graph for decoding. Similar to [2], we consider weighted RRD by cascading several WBP decoders.

For RRD, the computation of output LLRs is typically modified by introducing a scale factor before the sum in (6) [7, Eq. (4)]. We use a similar, but slightly different, approach. In particular, the soft input to the $\tau$-th decoder is modified to be a convex combination of the initial channel LLRs and the output LLRs of the $(\tau-1)$-th decoder according to

$$\boldsymbol{\ell}^{(\tau)} = \left[\beta\boldsymbol{\ell} + (1-\beta)\boldsymbol{m}^{((\tau-1)T_{\text{in}})}\right]^{\pi_\tau}, \tag{7}$$

where $\beta\in[0,1]$ is referred to as the mixing factor and $\boldsymbol{m}^{(0)} = \boldsymbol{\ell}$. Here, all BP messages and weights are iteration-indexed consecutively for $t\in[T]$, where $T = T_{\text{out}}T_{\text{in}}$. An example of the resulting feed-forward computation graph for RRD with $T_{\text{out}} = 2$ and $T_{\text{in}} = 2$ is shown in Fig. 1.

[1]Damping is referred to as "relaxed BP" in [2], where it is studied in the context of weighted min-sum decoding.

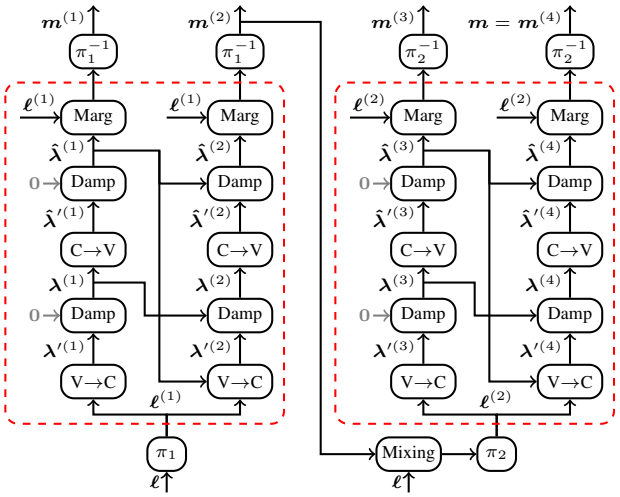



Fig. 1. Feed-forward computation graph for RRD with $T_{\text{out}} = T_{\text{in}} = 2$.

## III. Optimizing Weighted Belief-Propagation

It is well known that BP performs exact marginalization when the Tanner graph is a tree. However, good codes typically have loopy Tanner graphs with short cycles. To improve the BP performance, one can optimize the weights $w^{(t)}_{vc}$ and $w^{(t)}_v$ in all iterations [1]. The damping and mixing factors $\gamma$, $\beta$ can also be optimized. In the following, $\boldsymbol{o} = f(\boldsymbol{y};\theta)$ denotes the entire WBP mapping, where $\theta$ comprises all parameters.

### A. Deep Learning via Stochastic Gradient Descent

In [1], the authors propose to optimize $\theta$ using stochastic gradient descent (SGD) (or a variant thereof) based on many codeword–observation pairs $(\boldsymbol{x},\boldsymbol{y})$. In particular, the empirical loss $\mathcal{L}_A(\theta)$ for a finite set $A\subset\mathcal{C}\times\mathbb{R}^N$ of pairs is defined by

$$\mathcal{L}_A(\theta) \triangleq \frac{1}{|A|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in A} L\big(\boldsymbol{x}, f(\boldsymbol{y};\theta)\big), \tag{8}$$

where $L(\boldsymbol{a},\hat{\boldsymbol{a}})$ is the loss associated with returning the output $\hat{\boldsymbol{a}}$ when $\boldsymbol{a}$ is correct. Mini-batch SGD then uses the parameter update $\theta_{i+1} = \theta_i - \alpha\nabla_\theta\mathcal{L}_{B_i}(\theta_i)$, where $\alpha$ is the learning rate and $B_i$ is the mini-batch used in the $i$-th step. Due to channel and decoder symmetry, transmission of the all-zero codeword $\boldsymbol{x} = \boldsymbol{0}$ can be assumed for the optimization [1].

### B. Optimization Loss Function

For supervised classification problems, one typically uses cross-entropy loss. However, since the number of classes (i.e., codewords) scales exponentially with the block length, it is more practical to assume that the overall loss is the average of *bit-wise* losses according to

$$L(\boldsymbol{x},\boldsymbol{o}) = \frac{1}{N}\sum_{v=1}^{N} L_{\text{bit}}(x_v, o_v), \tag{9}$$

TABLE I
COMPARISON OF BIT-WISE LOSS FUNCTIONS.

| name | $L_{\text{bit}}(a,b)$ | $L_{\text{bit}}(0,b)$ |
|---|---|---|
| binary cross-entropy | $-\log(b^a(1-b)^{1-a})$ | $-\log(1-b)$ |
| negative soft bit success | $-b^a(1-b)^{1-a}$ | $-(1-b)$ |
| soft bit error | $(1-b)^a b^{1-a}$ | $b$ |

where $L_{\text{bit}}$ is a bit-wise loss function. For optimizing WBP, binary cross-entropy $L_{\text{bit}}(a,b) = -\log(b^a(1-b)^{1-a})$ is used in [1], [2]. However, our experiments show that minimizing (9) using binary cross entropy does not necessarily minimize the BER. To see why this may be the case, note that the negative bit success rate (per codeword) can be written as

$$\frac{1}{N}\sum_{v=1}^{N} L_{\text{bit}}(x_v, \hat{o}_v) = -\frac{1}{N}\sum_{v=1}^{N} \hat{o}_v^{x_v}(1-\hat{o}_v)^{1-x_v}, \qquad (10)$$

where $L_{\text{bit}}(a,b) = -b^a(1-b)^{1-a}$. On the other hand, inserting binary cross entropy into (9) leads to

$$L(\boldsymbol{x}, \boldsymbol{o}) = -\log\left(\prod_{v=1}^{N} o_v^{x_v}(1-o_v)^{1-x_v}\right)^{\frac{1}{N}}. \qquad (11)$$

Besides the log, the main difference between (10) and (11) is that arithmetic mean is used instead of geometric mean.

We propose a new loss function, where $L_{\text{bit}}(a,b) = (1-b)^a b^{1-a}$ is used in (9). This can be regarded as a "soft" version of BER since $(1-b)^a b^{1-a}$ for binary variables corresponds to $a$ XOR $b$, i.e., $L_{\text{bit}}(a,b)$ indicates a bit error. We refer to the resulting loss function as soft-BER. Tab. I summarizes the different binary loss functions and their simplification for the all-zero codeword. Also note that maximizing negative soft bit success is equivalent to minimizing soft bit error.

### C. Multi-Loss Optimization

The optimization behavior for WBP can be improved by using a multi-loss function [1], [2] (see also [4])

$$L(\boldsymbol{x}, \{\boldsymbol{o}^{(t)}\}_{t=1}^{T}) \triangleq \frac{1}{\sum_{t=1}^{T}\eta^{T-t}}\sum_{t=1}^{T}\eta^{T-t}L(\boldsymbol{x}, \boldsymbol{o}^{(t)}), \qquad (12)$$

where $\eta \in [0,1]$ is a discount factor. Multi-loss optimization takes into account the output after every iteration which helps to increase the magnitude of gradients corresponding to earlier iterations. We found that, rather than using a fixed discount factor as in [1], [2], [4], it is beneficial to decay $\eta$ during SGD, i.e., gradually moving from $\eta = 1$ (where the outputs of all BP iterations are considered with equal importance) towards $\eta = 0$ (where only the last BP iteration is taken into account).

### D. Weight Sharing

Excluding the damping/mixing factors, the total number of weights is $T(|E|+N)$ and we refer to this case as the fully-weighted (FW) decoder. In order to reduce the optimization complexity, one can share the weights, e.g., as follows:

- Temporal weight sharing (across decoding iterations), i.e.,
$$w_{vc}^{(t)} \equiv w_{vc}, \qquad w_v^{(t)} \equiv w_v, \qquad \forall\, t \in [T],$$
is referred to as RNN-FW, due to the similarity with recurrent neural networks (RNNs) [1].
- Spatial weight sharing (across edges), i.e.,
$$w_{vc}^{(t)} \equiv w_{\text{msg}}^{(t)}, \qquad w_v^{(t)} \equiv w_{\text{ch}}^{(t)}, \qquad \forall\,(v,c) \in E,$$
gives the simple-scaling (SS) model with two weights per iteration: one message and one channel weight.
- Temporal *and* spatial weight sharing gives two parameters in total. This is referred to as RNN-SS.

It is shown in [1], [2] that the RNN-FW structure gives similar gains as the FW decoder, i.e., there is little improvement when making parameters iteration-dependent. We further show that the RNN-SS structure incurs little to no performance penalty in many cases.

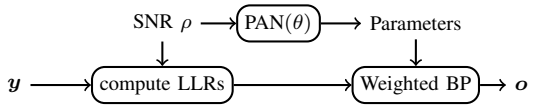


Fig. 2. Block diagram illustrating the parameter adapter network (PAN).

### E. Training SNR and Parameter Adapter Network

In general, the optimal WBP parameters may be different for different SNRs [2]. On the other hand, optimizing WBP separately for each SNR and storing the resulting weights is impractical if the set of possible SNRs is large or infinite. One general approach is to instead optimize assuming a range of different training SNRs [1], [2]. This leads to parameters that achieve a compromise between different channel conditions.

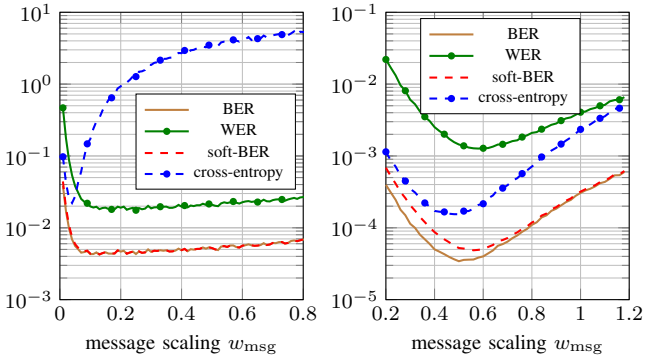
We propose a different approach where a PAN is used to learn the relation between the SNR $\rho$ and the corresponding optimal parameters.[2] Once trained, the PAN can be used to adaptively choose the best parameters for WBP corresponding to the channel conditions. The basic idea is illustrated in Fig. 2. In general, one can choose any structure to construct the PAN, e.g., a vanilla neural network. It is also possible to make only a subset of parameters SNR-adaptive.

In this paper, we use several shallow neural networks with one hidden layer of dimension 20 and output dimension 1 to model the SNR-dependency for each WBP parameter separately. ReLU activations are used for the hidden layer. The output layer uses sigmoid activations to ensure that the parameters satisfy their domain constraints, e.g., the damping factor is in the range $[0,1]$. For regular weights, we further scale the sigmoid outputs by 10 to increase the range to $[0,10]$. As an example, for WBP with the RNN-SS structure including damping, there are three parameters $w_{\text{msg}}$, $w_{\text{ch}}$, and $\gamma$. Thus, the PAN describes an SNR-parameterization according to $\text{PAN}(\rho) = [w_{\text{msg}}(\rho),\, w_{\text{ch}}(\rho),\, \gamma(\rho)] \in [0,10]^2 \times [0,1]$.

## IV. NUMERICAL RESULTS

The various decoding architectures in this paper are implemented in the PyTorch framework and optimized using the

[2]We assume perfect knowledge of the SNR. This knowledge is also required implicitly to compute channel LLRs. In practice, SNR is typically estimated and the SNR estimate can then be used as the input to the PAN.




(a) RM$(32, 16)$, overcomplete $\mathbf{H}_{\text{oc}}$ (b) BCH$(63, 36)$, cycle-reduced $\mathbf{H}_{\text{cr}}$

Fig. 3. Comparison of loss functions for RNN-SS with $w_{\text{ch}} = 1$, $\gamma = 0$, and $T = 3$. The SNR is $E_{\text{b}}/N_0 = 3\,\text{dB}$ in (a) and $E_{\text{b}}/N_0 = 7\,\text{dB}$ in (b).

RMSprop optimizer which is a variant of mini-batch SGD. Each mini-batch contains 100 observation pairs and the SNR for each pair is chosen from 10 equidistant points in the interval $[1\,\text{dB}, 8\,\text{dB}]$ such that exactly 10 pairs have the same SNR. The discount decay for the multi-loss optimization is implemented by starting with an initial discount factor $\eta = 1$ and multiplying $\eta$ by 0.5 after every 5000th SGD step. The same schedule is used to decay the learning rate, starting from $\alpha = 10^{-3}$ and using a decay rate of 0.8 instead of 0.5. To avoid numerical issues, a gradient clipping threshold of 0.1 is applied and the absolute values of the LLRs $\lambda_{v\to c}^{(t)}$ are clipped into the range $[-\log(\tanh(L_{\max}/2)), L_{\max}]$ with $L_{\max} = 15$.

The following Reed–Muller (RM) and Bose–Chaudhuri–Hocquenghem (BCH) codes are considered:

- RM$(32, 16)$ with standard PC matrix $\mathbf{H}_{\text{std}}$ (size $16 \times 32$) and overcomplete PC matrix $\mathbf{H}_{\text{oc}}$ ($620 \times 32$) whose rows are all minimum-weight dual codewords, see [9]
- BCH$(63, 36)$ with cycle-reduced PC matrix $\mathbf{H}_{\text{cr}}$ ($27 \times 63$) and right-regular PC matrix $\mathbf{H}_{\text{rr}}$ ($27 \times 63$), see [10]
- BCH$(127, 64)$ with cycle-reduced PC matrix $\mathbf{H}_{\text{cr}}$ ($63 \times 127$), see [10]

Ordered statistics decoding (OSD) is used as a benchmark whose performance is close to maximum-likelihood [11].

### A. Comparison of Loss Functions

We start by considering two RNN-SS structures with fixed $w_{\text{ch}} = 1$ and $\gamma = 0$ (i.e., no damping): (a) RM$(32, 16)$ with $\mathbf{H}_{\text{oc}}$ and (b) BCH$(63, 36)$ with $\mathbf{H}_{\text{cr}}$. The different loss functions for $T = 3$ are plotted in Fig. 3 as a function of $w_{\text{msg}}$, which is the only trainable parameter. For the RM code, cross-entropy has a sharp minimum at $w_{\text{msg}} \approx 0.05$, whereas soft-BER overlaps with BER and has a flat minimum at $w_{\text{msg}} \approx 0.15$. For the BCH code, the minima for cross-entropy, soft-BER, and BER all occur close to each other, but at slightly different locations.

In order to explain the distinct behavior of cross-entropy in Fig. 3(a), note that if a bit is decoded incorrectly, binary cross-entropy gives a penalty close to the magnitude of the output LLR $|m_v|$. This is problematic in cases where the decoder is wrong, but very sure about its decision. Indeed, this behavior

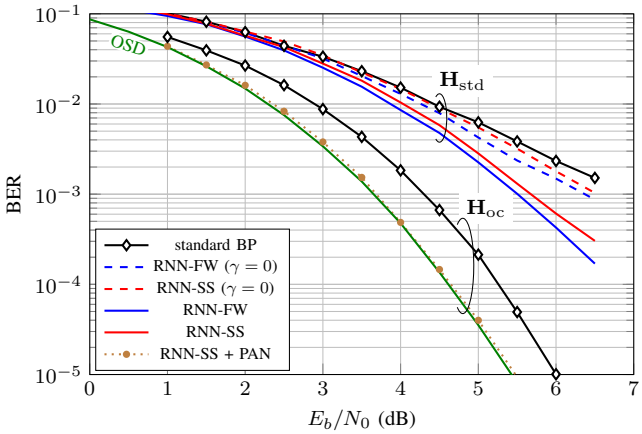



Fig. 4. Results for RM$(32, 16)$ with $\mathbf{H}_{\text{oc}}$ ($T = 5$) and $\mathbf{H}_{\text{std}}$ ($T = 20$).

is characteristic for BP with highly redundant PC matrices and such failure cases tend to dominate the average loss. This effect is even more pronounced for large $T$ since the average LLR magnitude tends to grow with the iteration number.

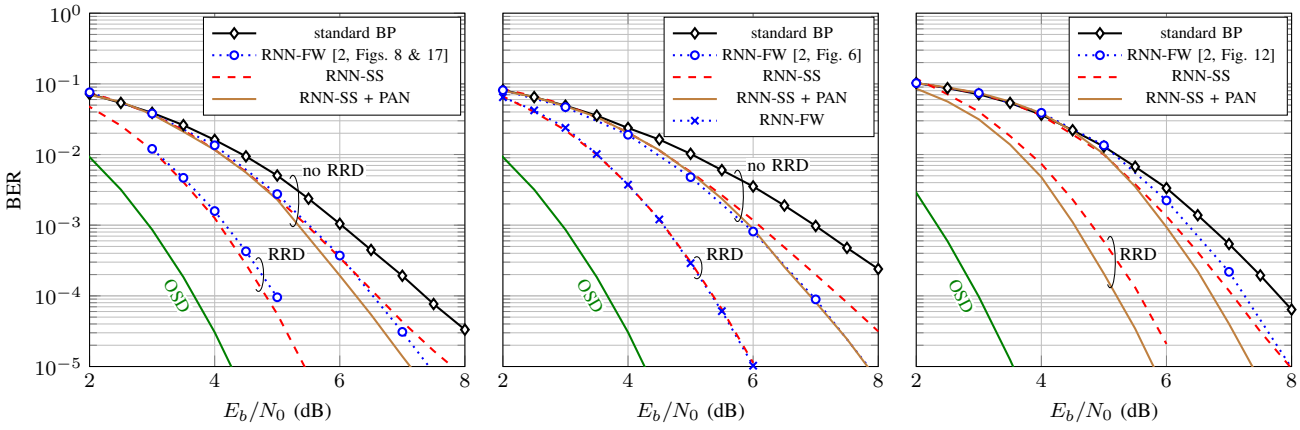
The results in Fig. 3 show that, in general, neither cross-entropy nor soft-BER are guaranteed to minimize BER. All scenarios in this paper were optimized using both functions. We found that they give comparable results, with the exception of highly redundant PC matrices where soft-BER is preferable.

### B. Reed–Muller Codes

Results for RM$(32, 16)$ assuming both RNN-FW and RNN-SS structures are shown in Fig. 4. For $\mathbf{H}_{\text{std}}$ with $T = 20$ iterations, simple scaling results in a performance loss of up to 0.3 dB. Damping gives considerable performance improvements in both cases, at the expense of additional computational complexity and storage requirements. For $\mathbf{H}_{\text{oc}}$ with $T = 5$, the RNN-SS structure is sufficient to achieve close-to-optimal performance and the overlapping results for RNN-FW are omitted. For this case, we note that the optimization with soft-BER gives lower BER than for CE, as expected from the discussion in the previous subsection.

### C. BCH Codes

For the BCH codes, the parameters are chosen to facilitate a direct comparison with [2]. In particular, we fix $T = 5$ and $\gamma = 0$, i.e., no damping is used. Results for BCH$(63, 36)$ with $\mathbf{H}_{\text{cr}}$ are shown in Fig. 5(a), where we compare with the best results in [2, Fig. 8] for the same parameters. The RNN-SS with two trainable parameters achieves similar gains as the RNN-FW in [2] for BERs $> 10^{-4}$. For lower BERs, the performance starts to deviate. The situation can be improved by making the parameters SNR-adaptive using the proposed PAN approach. In this case, the performance improves markedly for high SNRs. This is due to the fact that training over a range of SNRs without the PAN tends to focus almost exclusively on low-SNR/high-BER regions. Similar observations can be made for the same code with $\mathbf{H}_{\text{rr}}$, as shown in Fig. 5(b). In this case, RNN-FW performs slightly better than RNN-SS for some SNRs, i.e., two parameters are not sufficient to obtain




(a) BCH$(63, 36)$, cycle-reduced $\mathbf{H}_{\text{cr}}$ (b) BCH$(63, 36)$, right-regular $\mathbf{H}_{\text{rr}}$ (c) BCH$(127, 64)$, cycle-reduced $\mathbf{H}_{\text{cr}}$

Fig. 5. Results for BCH codes assuming $\gamma = 0$ (i.e., no damping), $T = 5$ iterations (no RRD) and $T_{\text{in}} = 2$, $T_{\text{out}} = 30$ for RRD. Data points are extracted directly from the relevant figures in [2]. Note that our results for BCH$(127, 64)$ are not directly comparable to [2] because of potentially different PC matrices.

the full gain. Finally, results for BCH$(127, 64)$ with $\mathbf{H}_{\text{cr}}$ are shown in Fig. 5(c). We caution the reader that these results are not directly comparable because our standard BP performs better than what is shown in [2, Fig. 12]. This is likely due to different cycle-reduced PC matrices. However, we were not able to improve upon the shown RNN-SS results using RNN-FW, which indicates that the simple-scaling approach is also sufficient in this case.

We also consider weighted RRD for BCH$(63, 36)$ and BCH$(127, 64)$ where $T_{\text{in}} = 2$, $T_{\text{out}} = 30$, and the mixing factor is treated as an additional optimization parameter. Results for $\mathbf{H}_{\text{cr}}$ are shown in Fig. 5(a) and we compare to the corresponding results in [2, Fig. 12] labeled as "mRRD-RNN(1)". We obtain slightly better performance using RNN-SS even without a PAN. This can be attributed to the improved training methodology, particularly the discount decay for the multi-loss optimization. For $\mathbf{H}_{\text{rr}}$, no RRD results are available in [2] and we compare to our own results. Both RNN-FW and RNN-SS give virtually the same performance as shown in Fig. 5(b). Fig. 5(c) shows that the PAN also gives some extra performance gain for RRD decoding. Additional simulation results for larger $T$ can be found in [12].

## V. Discussion and Conclusion

In this paper, we have considered WBP decoding of short Reed–Muller and BCH codes. Our experiments support the observations in [1], [2] that optimizing WBP can provide meaningful gains. In addition, we have shown that simple-scaling models with fewer parameters are often sufficient to achieve gains similar to the full parameterization. This can lead to a considerably simpler optimization procedure and greatly reduce complexity, e.g., in terms of storage requirements. In general, the performance loss incurred by simple scaling depends on the employed PC matrix. Small penalties were observed for matrices with highly irregular degree distributions (e.g., $\mathbf{H}_{\text{std}}$ for RM(32,16) or $\mathbf{H}_{\text{rr}}$ for BCH(63, 36)), whereas the loss appears to be negligible if the degree distribution is regular ($\mathbf{H}_{\text{oc}}$ for RM(32,16)) or RRD is employed.

It was also shown that choosing a suitable loss function for the optimization is scenario-dependent. For highly redundant PC matrices, it was found that binary cross-entropy penalizes too hard on bit errors where the decoder is very sure about its decision. In such cases, optimizing with the proposed soft-BER loss leads to better performance. Lastly, we built on the observation in [2] that the optimal WBP parameters are SNR-dependent and proposed a simple solution based on parameter adapter networks. This approach allows us to learn optimal parameters for multiple SNRs in a single training process without trading off performance between channel conditions.

## References


[1] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *Proc. Annual Allerton Conference on Communication, Control, and Computing*, Illinois, USA, 2016.

[2] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be'ery, "Deep learning methods for improved decoding of linear codes," *IEEE J. Sel. Topics Signal Proc.*, vol. 12, no. 1, pp. 119–131, Feb. 2018.

[3] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," in *Proc. Annual Conf. Information Sciences and Systems (CISS)*, 2017.

[4] A. Bennatan, Y. Choukroun, and P. Kisilev, "Deep learning for decoding of linear codes - a syndrome-based approach," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Vail, CO, 2018.

[5] M. Fossorier, R. Palanki, and J. Yedidia, "Iterative decoding of multi-step majority logic decodable codes," in *Proc. Int. Symp. on Turbo Codes & Iterative Inform. Proc.*, 2003, pp. 125–132.

[6] J. S. Yedidia, J. Chen, and M. P. Fossorier, "Generating code representations suitable for belief propagation decoding," in *Proc. Annual Allerton Conf. on Commun., Control, and Comp.*, vol. 40, no. 1, 2002, pp. 447–456.

[7] J. Jiang and K. R. Narayanan, "Iterative soft decoding of Reed-Solomon codes," *IEEE Commun. Lett.*, vol. 8, no. 4, pp. 244–246, April 2004.

[8] T. R. Halford and K. M. Chugg, "Random redundant soft-in soft-out decoding of linear block codes," in *Proc. IEEE Int. Symp. Inform. Theory*. IEEE, 2006, pp. 2230–2234.

[9] E. Santi, C. Häger, and H. D. Pfister, "Decoding Reed-Muller codes using minimum-weight parity checks," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Vail, CO, 2018.

[10] M. Helmling, S. Scholl, F. Gensheimer, T. Dietz, K. Kraft, S. Ruzika, and N. Wehn, "Database of channel codes and ML simulation results," *www.uni-kl.de/channel-codes*, 2017.

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

[12] M. Lian, C. Häger, and H. D. Pfister, "What can machine learning teach us about communications?" in *Proc. IEEE Information Theory Workshop (ITW)*, Guangzhou, China, 2018.