DEEP EXTENDED FEEDBACK CODES

Anahid Robert Safavi[1], Alberto G. Perotti[1], Branislav M. Popović[1], Mahdi Boloursaz Mashhadi[2], Deniz Gündüz[2]
[1]Radio Transmission Technology Laboratory, Huawei Technologies Sweden AB, Kista 164-94, Sweden,
[2]Information Processing and Communications Laboratory, Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2BT, U.K.

NOTES: Corresponding author: Alberto G. Perotti, alberto.perotti@huawei.com

Anahid Robert Safavi is now with the Wireless Network Algorithm Laboratory, Huawei Technologies Sweden AB.

Abstract *– A new Deep Neural Network (DNN)-based error correction encoder architecture for channels with feedback, called Deep Extended Feedback (DEF), is presented in this paper. The encoder in the DEF architecture transmits an information message followed by a sequence of parity symbols which are generated based on the message as well as the observations of the past forward channel outputs sent to the transmitter through a feedback channel. DEF codes generalize Deepcode [1] in several ways: parity symbols are generated based on forward channel output observations over longer time intervals in order to provide better error correction capability; and high-order modulation formats are deployed in the encoder so as to achieve increased spectral efficiency. Performance evaluations show that DEF codes have better performance compared to other DNN-based codes for channels with feedback.*

Keywords – Deep learning, error correction, feedback, ultra-reliable

1. INTRODUCTION

The fifth generation (5G) wireless cellular networks' New Radio (NR) access technology has been recently specified by the 3rd Generation Partnership Project (3GPP). NR already fulfills demanding requirements of throughput, reliability and latency. However, new use cases stemming from new application domains (such as industrial automation, vehicular communications or medical applications) call for further significant enhancements. For instance, some typical Industrial Internet of Things (IIoT) applications would need considerably higher reliability and shorter transmission delay compared to what 5G/NR can provide nowadays.

Error correction coding is a key physical layer functionality for guaranteeing the required performance levels. In conventional systems, error correction is accomplished by linear binary codes such as polar codes [2], Low Density Parity Check (LDPC) codes [3] or turbo codes [4], possibly combined with retransmission mechanisms such as Hybrid Automatic Request (HARQ) [5]. HARQ performs an initial transmission followed by a variable number of subsequent incremental redundancy transmissions until the receiver notifies successful decoding to the transmitter. Short Acknowledgment (ACK) or Negative ACK (NACK) messages are sent through a feedback channel in order to inform the transmitter about decoding success. By usage of simple ACK/NACK feedback messages, conventional HARQ practically limits the gains that could potentially be obtained by an extensive and more efficient use of the feedback channel. Codes that make full use of feedback potentially achieve improved performance compared to conventional codes, as predicted in [6].

Finding good codes for channels with feedback is a notoriously difficult problem. Several coding methods for channels with feedback have been proposed; see for example [7,8,9,10,11]. However, all known solutions either do not approach the performance predicted in [6] or exhibit unaffordable complexity. Promising progress has been made recently by applying Machine Learning (ML) methods [1], where both encoder and decoder are implemented as two separate Deep Neural Networks (DNNs). The DNNs' coefficients are determined through a joint encoder-decoder training procedure whereby encoder and decoder influence each other. In that sense, the chosen *decoder* structure has impact on the resulting code – a previously unseen feature. Known DNN-based feedback codes [1] use different recurrent Neural Network (NN) architectures, Recurrent NNs (RNNs) and Gated Recurrent Units (GRUs) are used in [1]; Long-Short Term Memory (LSTM) architectures have been mentioned in a preprint of [1] as a potential alternative to RNNs for the encoder.

A new DNN-based code for channels with feedback called Deep Extended Feedback (DEF) code is presented in this paper. The encoder transmits an information message followed by a sequence of parity symbols which are generated based on the message and on observations of the past forward channel outputs obtained through the feedback channel. Known DNN-based codes for channels with feedback [1] compute their parity symbols based on the information message and on the most recent information received through the feedback channel. The DEF code is based on *feedback extension*, which consists of extending the encoder input so as to comprise delayed versions of feedback signals. Thus, the DEF encoder input comprises the most recent feedback signal and a set of past feedback signals within a given time window. A similar

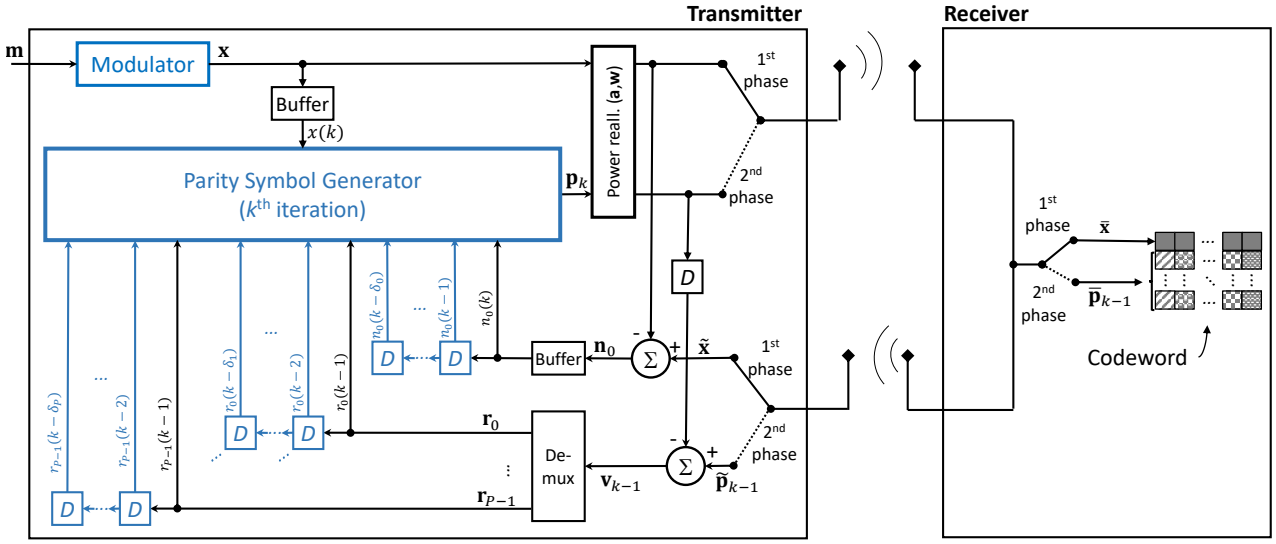

Fig. 1 – DEF encoder structure. Each "D" block represents a unit-time delay. Blue blocks and signals denote new functionalities compared to prior solutions.

approach could be used in the decoder to extend its input so as to comprise delayed versions of received signals in a given time window. However, it can be shown that such a generalization of the decoder does not bring any benefit and therefore it will not be considered in the definition of DEF codes. The extended-feedback encoder architecture is combined with different NN architectures of recurrent type, namely RNN, GRU and LSTM. The DEF code generalizes Deepcode [1] along several directions. Its major benefits can be summarized as follows:

- **Improved error correction capability obtained by feedback extension**. The DEF code generates parity symbols based on feedbacks in a longer time window, thereby introducing long-range dependencies between parity symbols. As the above long-range dependencies are a necessary ingredient of all good error correcting codes, it is expected that feedback extension will bring performance improvements.
- **Higher spectral efficiency obtained by usage of QAM/PAM modulations**. The DEF code uses Quadrature Amplitude Modulation (QAM) with arbitrary order, thereby potentially achieving higher spectral efficiency.

In this work, we initially focus on DEF codes' performance evaluation over channels with *noiseless feedback*, where the forward-channel output observations are sent uncorrupted to the encoder.

Notation: Lower case and upper case letters denote scalar (real or complex) values. For any pair of positive integers a and b with $a < b$, $[a : b]$ denotes the sequence of integers $[a, a+1, \dots, b]$, sorted in increasing order. Boldface lower case letters (e.g., $\mathbf{b}$) denote vectors; unless otherwise specified, all vectors are assumed to be column vectors. $b(i)$ denotes the i^{th} element of $\mathbf{b}$; $\mathbf{b}(j:k), j < k$, denotes the sub-vector that contains the elements of $\mathbf{b}$ with indices in $[j : k]$. Boldface upper case letters like $\mathbf{A}$ denote matrices; $a_{i,j}$ represents the element of $\mathbf{A}$ in the i^{th} row and j^{th} column. Notation $f(\mathbf{v})$, where f is a function taking a scalar input, indicates the vector obtained by applying f to each element of $\mathbf{v}$. Hadamard (i.e., element-wise) product is denoted by $\circ$.

2. DEFINITION OF DEEP EXTENDED FEEDBACK CODE

The Deep Extended Feedback (DEF) code is the set of codewords produced by the DEF encoder shown in Fig. 1. Blue blocks and signals in Fig. 1 denote the new functionalities of the DEF code compared to Deepcode [1], *extended feedback* is shown by the unit-time delay blocks labeled "D" and their corresponding input/ouput signals; QAM/PAM symbols are produced by the block labeled "Modulator". DEF code and Deepcode operate according to the same encoding procedure as described later on. The novel DEF code features will be treated in dedicated subsections.

The encoding procedure consists of two phases. In the *first phase*, an L-bit information message $\mathbf{m} = (m(0), \dots, m(L-1))$ is mapped to a sequence of real symbols $\mathbf{x} = (x(0), \dots, x(K-1))$, hereafter called *systematic* symbols.

The modulation sequence $\mathbf{x}$ is transmitted on the forward channel. The corresponding sequence $\bar{\mathbf{x}}$ observed by the receiver is given by

$$\bar{\mathbf{x}} = \mathbf{x} + \mathbf{n}_0 \qquad (1)$$

where $\mathbf{n}_0$ represents Additive White Gaussian Noise (AWGN) and other possible forward-channel impairments. In the performance evaluations of Section 4, $\mathbf{n}_0$ is modeled as a sequence of white Gaussian noise samples.

The receiver stores the observed signal $\bar{\mathbf{x}}$ locally and immediately echoes it back to the transmitter through the feedback channel. A corresponding sequence

$$\tilde{\mathbf{x}} = \bar{\mathbf{x}} + \mathbf{g}_0 \quad (2)$$

is obtained at the transmitter, where $\mathbf{g}_0$ represents additive white Gaussian noise and other possible feedback-channel impairments.
In the *second phase*, for each element $x(k)$ of $\mathbf{x}$, the encoder computes a corresponding sequence of parity symbols

$$\mathbf{p}_k = (p_k(0), \dots, p_k(P-1)), \quad k = 0, \dots, K-1 \quad (3)$$

and transmits it through the forward channel. P is the number of parity symbols that the encoder generates per systematic symbol. Thus, the total number of transmitted symbols is $K(1+P)$. The DEF code rate is defined as the ratio of the message length L over $K(1+P)$, that is:

$$R_{\text{DEF}} \triangleq \frac{L}{K(1+P)}. \quad (4)$$

The receiver observes a set of corresponding parity symbols sequences $\bar{\mathbf{p}}_k, k = 0, \dots, K-1$. $\bar{\mathbf{p}}_k$ can be written as follows:

$$\bar{\mathbf{p}}_k = \mathbf{p}_k + \mathbf{v}_k, \quad (5)$$

where $\mathbf{v}_k = (v_k(0), \dots, v_k(P-1))$ represents additive white Gaussian noise and other forward channel impairments. $\bar{\mathbf{p}}_k$ is immediately echoed back to the transmitter through the feedback channel so as to obtain

$$\tilde{\mathbf{p}}_k = \bar{\mathbf{p}}_k + \mathbf{g}_k, \quad (6)$$

where $\mathbf{g}_k$ represents additive white Gaussian noise and other feedback channel impairments.
The DEF codeword is defined as $\mathbf{z} = (z(0), \dots, z((P+1)K-1))$. The j^{th} codeword symbol is defined as follows:

$$\begin{aligned} z(j) &= \begin{cases} w(0)a(j)x(j), & 0 \le j \le K-1 \\ w(l+1)a(k)p_k(l), & K \le j \le (P+1)K-1 \end{cases} \quad (7) \\ l &= (j-K) \bmod P, \\ k &= \lfloor (j-K)/P \rfloor, \end{aligned}$$

where $w(0)$ and $w(l+1), l = 0, \dots, P-1$, are *codeword* power levels, $a(k), k = 0, \dots, K-1$, are *symbol* power levels, $x(j)$ is the j^{th} systematic symbol, and $p_k(l)$ is the l^{th} symbol of the k^{th} parity sequence *(3)*. Codeword power levels reallocate the power among codeword symbols as follows: the systematic symbols are scaled by $w(0)$; the 1^{st} parity symbol of each parity sequence is scaled by $w(1)$, the 2^{nd} parity symbol of each parity sequence is scaled by $w(2)$, etc. Symbol power levels reallocate the power among codeword symbols as follows: $a(0)$ scales the amplitude of the 1^{st} systematic symbol $x(0)$ and of the symbols of the 1^{st} parity symbol sequence $\mathbf{p}_0$, $a(1)$ scales the amplitude of the 2^{nd} systematic symbol $x(1)$ and of the symbols of the 2^{nd} parity symbol sequence $\mathbf{p}_1$, ... $a(K-1)$ scales the amplitude of the K^{th} systematic symbol $x(K-1)$ and of the symbols of the K^{th} parity symbol sequence $\mathbf{p}_{K-1}$. Power levels $w(l)$ and $a(k)$ are obtained by NN training. The following constraints preserve the codeword's average power:

$$\sum_{l=0}^{P} w^2(l) = 1, \quad \sum_{k=0}^{K-1} a^2(k) = 1. \quad (8)$$

2.1 QAM/PAM modulator

The DEF code modulator maps the L-bit information message $\mathbf{m} = (m(0), \dots, m(L-1))$ to a sequence of real symbols $\mathbf{x} = (x(0), \dots, x(K-1))$, hereafter called *systematic* symbols. Each pair of consecutive symbols $(x(2i), x(2i+1)), i = 0, \dots, K/2-1$, forms a complex QAM symbol $q(i) = x(2i) + x(2i+1)\sqrt{-1}$, where $q(i)$ is obtained by mapping Q consecutive bits of $\mathbf{m}$ to 2^Q-QAM. The above mapping produces $K = 2L/Q$ real systematic symbols at the modulator output.
Examples of QAM/PAM mapping of order $Q = 2$ and $Q = 4$ are shown in Table 1 and Table 2.

2.2 Extended feedback

We call *Parity Symbol Generator* (PSG) the encoder block that computes the parity symbol sequences (see Fig. 1). Extended feedback consists of sending to the PSG a sequence of forward-channel output observations over longer time intervals compared to Deepcode [1].

Table 1 – Example of QAM/PAM mapping of order $Q = 2$.

$m(2i), m(2i+1)$	$x(2i)$	$x(2i+1)$
$0, 0$	1	1
$0, 1$	1	-1
$1, 0$	-1	1
$1, 1$	-1	-1

Table 2 – Example of QAM/PAM mapping of order $Q = 4$.

$m(4i), m(4i+1), m(4i+2), m(4i+3)$	$x(2i), x(2i+1)$
$0, 0, 0, 0$	3, 3
$0, 0, 0, 1$	3, 1
$0, 0, 1, 0$	3, -3
$0, 0, 1, 1$	3, -1
$0, 1, 0, 0$	1, 3
$0, 1, 0, 1$	1, 1
$0, 1, 1, 0$	-1, -3
$0, 1, 1, 1$	-1, -1
$1, 0, 0, 0$	-3, 3
$1, 0, 0, 1$	-3, 1
$1, 0, 1, 0$	-3, -3
$1, 0, 1, 1$	-3, -1
$1, 1, 0, 0$	-1, 3
$1, 1, 0, 1$	-1, 1
$1, 1, 1, 0$	-1, -3
$1, 1, 1, 1$	-1, -1

The PSG input column vector at the k^{th} iteration is defined as follows:

$$\mathbf{i}_k = \begin{bmatrix} x(k) \\ \mathbf{n}_0(k-\delta_0 : k) \\ \mathbf{r}_0(k-\delta_1 : k-1) \\ \cdots \\ \mathbf{r}_{P-1}(k-\delta_P : k-1) \end{bmatrix}, \qquad (9)$$

where $x(k)$ is the k^{th} systematic symbol, $\mathbf{n}_0(k-\delta_0 : k)$ is a column vector of length δ_0+1 which contains noise samples from the sequence $\mathbf{n}_0$ of *(1)*, $\mathbf{r}_l(k-\delta_l : k-1)$ $(l=0,\ldots,P-1)$ is a column vector of length δ_l which contains noise samples from the sequence $\mathbf{r}_l$ of forward-channel noise samples that corrupt the l^{th} symbol of each parity symbol sequence, that is:

$$\mathbf{r}_l \triangleq (v_0(l),\ldots,v_{K-1}(l)), \qquad (10)$$

where $v_k(l)$ $(k=0,\ldots,K-1)$ is the l^{th} sample of $\mathbf{v}_k$ in *(5)* and $\delta_0,\ldots,\delta_P$ are arbitrary positive integers (δ_0 can be 0), hereafter called the *encoder input extensions*. We note that the Deepcode [1] encoder can be recovered as a special case by setting $\delta_0 = 0$ and $\delta_1 = \ldots = \delta_P = 1$, which means that, in each iteration, only a single noise sample for each systematic or parity check symbol is used. The buffers in the DEF encoder contain the systematic symbol sequence $\mathbf{x}$ and the corresponding forward-noise sequence $\mathbf{n}_0$ of *(1)*. Those sequences are generated during the first encoding phase and used by the PSG in the second phase.

2.3 Parity Symbol Generator (PSG)

The core functionality of the DEF encoder is the computation of the parity check symbols, which is performed by the block denoted (PSG) (see Fig. 1). PSG computes the k^{th} parity symbol sequence $\mathbf{p}_k$ based on the k^{th} modulation symbol x_k and a subset of the past forward-channel outputs.

Fig. 2 shows the structure of the PSG. In the k^{th} encoding iteration, the PSG generates a k^{th} parity symbol sequence $\mathbf{p}_k$ which consists of P real parity symbols obtained as follows:

$$\mathbf{p}_k = \text{Norm}(e(\mathbf{h}_k)), \qquad (11)$$

where $\mathbf{h}_k$, a real vector of arbitrary length H_0, denotes the PSG state at time instant k, while function $e(\cdot)$ consists of a linear transformation applied to the PSG state $\mathbf{h}_k$ obtained as follows:

$$e(\mathbf{h}_k) = \mathbf{A}\mathbf{h}_k + \mathbf{c}, \qquad (12)$$

where $\mathbf{A}$ has size $P \times H_0$ and $\mathbf{c}$ has length P. The above matrices $\mathbf{W}$, $\mathbf{Y}$, $\mathbf{A}$ and vectors $\mathbf{b}$, $\mathbf{c}$ are obtained by NN training. The $\text{Norm}(\cdot)$ function normalizes the PSG output so that each parity symbol has zero mean and unit variance. The PSG state $\mathbf{h}_k$ is recursively computed as

$$\mathbf{h}_k = f(\mathbf{i}_k, \mathbf{h}_{k-1}), \qquad (13)$$

where function $f(\cdot)$ will be discussed below, and $\mathbf{i}_k$ is defined in *(9)*. As for the initialization, we set $\mathbf{h}_0$ as the all-zero vector.

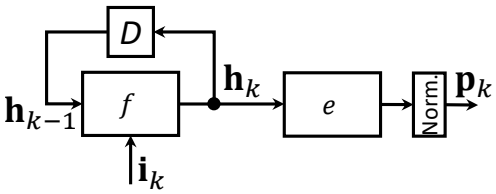

Fig. 2 – Structure of the PSG.

Functions e and f will be parameterized using DNNs. The structure of Fig. 2 corresponds to a recurrent architecture, and therefore, we will consider the following three recurrent architectures to model it: RNNs, GRUs and LSTMs.

2.3.1 RNN

When modeled with an RNN, the function $f(\cdot)$ in *(13)* is defined as follows:

$$f(\mathbf{i}_k, \mathbf{h}_{k-1}) = \tanh(\mathbf{W}\mathbf{h}_{k-1} + \mathbf{Y}\mathbf{i}_k + \mathbf{b}), \qquad (14)$$

where $\mathbf{W}$ is a *state-transition matrix* of size $H_0 \times H_0$, $\mathbf{Y}$ is an *input-state matrix* of size $H_0 \times I$ (I is the length of vector $\mathbf{i}_k$), and $\mathbf{b}$ is a *bias vector* of length H_0. $\mathbf{W}$, $\mathbf{Y}$ and $\mathbf{b}$ are obtained by NN training.

2.3.2 GRU

With a GRU, the function $f(\cdot)$ of *(13)* is defined as follows:

$$\begin{aligned} f(\mathbf{i}_k, \mathbf{h}_{k-1}) &= f_0(\mathbf{i}_k, \mathbf{h}_{k-1}) \circ (1 - z(\mathbf{i}_k, \mathbf{h}_{k-1})) \\ &+ \mathbf{h}_{k-1} \circ z(\mathbf{i}_k, \mathbf{h}_{k-1}). \end{aligned} \qquad (15)$$

The function $f_0(\cdot)$ in *(15)* is defined as follows:

$$\begin{aligned} f_0(\mathbf{i}_k, \mathbf{h}_{k-1}) &= \tanh((\mathbf{W}_f \mathbf{h}_{k-1} + \mathbf{b}_h) \circ r(\mathbf{i}_k, \mathbf{h}_{k-1}) \\ &+ \mathbf{Y}_f \mathbf{i}_k + \mathbf{b}_i). \end{aligned} \qquad (16)$$

The functions $z(\cdot)$ in *(15)* and $r(\cdot)$ in *(16)* are defined as follows:

$$z(\mathbf{i}_k, \mathbf{h}_{k-1}) = \sigma(\mathbf{W}_z \mathbf{h}_{k-1} + \mathbf{Y}_z \mathbf{i}_k + \mathbf{b}_z) \qquad (17)$$

$$r(\mathbf{i}_k, \mathbf{h}_{k-1}) = \sigma(\mathbf{W}_r \mathbf{h}_{k-1} + \mathbf{Y}_r \mathbf{i}_k + \mathbf{b}_r) \qquad (18)$$

where $\sigma(x) \triangleq (1+e^{-x})^{-1}$ denotes the *sigmoid* function. In equations *(15)-(18)*, matrices $\mathbf{W}_f, \mathbf{W}_z, \mathbf{W}_r, \mathbf{Y}_f, \mathbf{Y}_z, \mathbf{Y}_r$ and vectors $\mathbf{b}_h, \mathbf{b}_i, \mathbf{b}_z, \mathbf{b}_r$ are obtained by NN training.

2.3.3 LSTM

As for LSTM, the function $f(\cdot)$ of *(13)* is defined as follows:

$$f(\mathbf{i}_k, \mathbf{h}_{k-1}) = f_1(\mathbf{i}_k, \mathbf{h}_{k-1}) \circ \tanh(\mathbf{s}_k) \qquad (19)$$

where $\mathbf{s}_k$ is the *cell state* at time instant k. The cell state provides long-term memory capability to the LSTM NN, whereas the state $\mathbf{h}_k$ provides short-term memory capability. The cell state is recursively computed as follows:

$$\begin{aligned} \mathbf{s}_k &= f_2(\mathbf{i}_k, \mathbf{h}_{k-1}) \circ \mathbf{s}_{k-1} \\ &+ f_3(\mathbf{i}_k, \mathbf{h}_{k-1}) \circ f_4(\mathbf{i}_k, \mathbf{h}_{k-1}). \end{aligned} \qquad (20)$$

The function f_1 in *(19)* and functions f_2, f_3 and f_4 in *(20)* are defined as follows:

$$f_1(\mathbf{i}_k, \mathbf{h}_{k-1}) = \sigma(\mathbf{W}_1\mathbf{h}_{k-1} + \mathbf{Y}_1\mathbf{i}_k + \mathbf{b}_1) \quad (21)$$
$$f_2(\mathbf{i}_k, \mathbf{h}_{k-1}) = \sigma(\mathbf{W}_2\mathbf{h}_{k-1} + \mathbf{Y}_2\mathbf{i}_k + \mathbf{b}_2) \quad (22)$$
$$f_3(\mathbf{i}_k, \mathbf{h}_{k-1}) = \sigma(\mathbf{W}_3\mathbf{h}_{k-1} + \mathbf{Y}_3\mathbf{i}_k + \mathbf{b}_3) \quad (23)$$
$$f_4(\mathbf{i}_k, \mathbf{h}_{k-1}) = \tanh(\mathbf{W}_4\mathbf{h}_{k-1} + \mathbf{Y}_4\mathbf{i}_k + \mathbf{b}_4) \quad (24)$$

In equations *(21)-(24)*, matrices $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_3$, $\mathbf{W}_4$, $\mathbf{Y}_1$, $\mathbf{Y}_2$, $\mathbf{Y}_3$, $\mathbf{Y}_4$ and vectors $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$, $\mathbf{b}_4$ are obtained by NN training.

2.4 Mitigation of unequal bit error distribution

It has been observed in [1] that the feedback codes based on RNNs exhibit a non-uniform bit error distribution, i.e., the final message bits typically have a significantly larger error rate compared to other bits. In order to mitigate the detrimental effect of non-uniform bit error distribution, [1] introduced two countermeasures:

- *Zero-padding*. Zero-padding consists in appending at least one information bit with predefined value (e.g., zero) at the end of the message. The appended information bit(s) are discarded at the decoder, such that the positions affected by higher error rates carry no information.
- *Power reallocation*. Zero-padding alone is not enough to mitigate unequal errors, and moreover it reduces the effective code rate. Instead, power reallocation redistributes the power among the codeword symbols so as to provide better error protection to the message bits whose positions are more error-prone, i.e., the initial and final positions.

2.5 DEF decoder

In DNN-based codes, encoder and decoder are implemented as two separate DNNs whose coefficients are determined through a joint encoder-decoder training procedure. Therefore, the *encoder* structure has impact on the *decoder* coefficients obtained through training, and vice-versa. In that sense, the chosen *decoder* structure has impact on the resulting code.

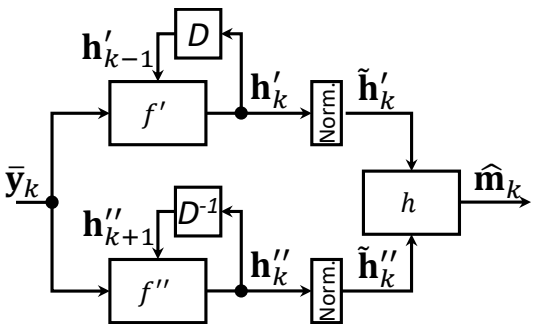

Fig. 3 – DEF decoder.

The DEF decoder (see Fig. 3) maps the received DEF codeword to a decoded message $\hat{\mathbf{m}}$ as follows:

$$\hat{\mathbf{m}} = g(\bar{\mathbf{x}}, \bar{\mathbf{p}}^{(1)}, \dots, \bar{\mathbf{p}}^{(K)}). \quad (25)$$

The decoder consists of a bidirectional recurrent NN (a GRU or LSTM) followed by a linear transformation and a sigmoid function. The bidirectional recurrent NN computes a sequence of forward-states $\mathbf{h}'_k$ and backward-states $\mathbf{h}''_k$ as follows:

$$\mathbf{h}'_k = f'(\bar{\mathbf{y}}_k, \mathbf{h}'_{k-1}) \quad (26)$$
$$\mathbf{h}''_{k-1} = f''(\bar{\mathbf{y}}_k, \mathbf{h}''_k) \quad (27)$$

where functions f', f'' are defined as in *(15)* for the GRU-based decoder and as in *(19)* for the LSTM-based decoder, and the input column vector $\bar{\mathbf{y}}_k$ is defined as follows:

$$\bar{\mathbf{y}}_k = \begin{bmatrix} \bar{\mathbf{x}}(k-\gamma_0 : k) \\ \bar{\mathbf{q}}_0(k-\gamma_1 : k) \\ \dots \\ \bar{\mathbf{q}}_{P-1}(k-\gamma_P : k) \end{bmatrix}, \quad (28)$$

where $\bar{\mathbf{x}}(k-\gamma_0 : k)$ is a column vector of length $\gamma_0 + 1$ which contains symbols from the received systematic sequence $\bar{\mathbf{x}}$ of *(1)*, and $\bar{\mathbf{q}}_l(k-\gamma_l : k), l = 0, \dots, P-1$, is a column vector of length $\gamma_l + 1$ containing symbols from the sequence $\bar{\mathbf{q}}_l$, which consists of the l^{th} symbol of each received parity sequence $\bar{\mathbf{p}}_k$ *(5)*. $\bar{\mathbf{q}}_l$ is defined as follows:

$$\bar{\mathbf{q}}_l \triangleq (\bar{p}_0(l), \dots, \bar{p}_{K-1}(l)), l = 0, \dots, P-1. \quad (29)$$

Finally, the values $\gamma_0, \dots, \gamma_P$ are arbitrary non-negative integers, hereafter called the *decoder input extensions*. The initial forward NN state $\mathbf{h}'_0$ and the initial backward NN state $\mathbf{h}''_K$ are set as all-zero vectors.
The k^{th} decoder output is obtained as follows:

$$\hat{\mathbf{m}}_k = h(\tilde{\mathbf{h}}'_k, \tilde{\mathbf{h}}''_{k-1}) \triangleq \sigma\left(\mathbf{C}\begin{bmatrix} \tilde{\mathbf{h}}'_k \\ \tilde{\mathbf{h}}''_{k-1} \end{bmatrix} + \mathbf{d}\right), \quad (30)$$

where $\sigma(\cdot)$ is the *sigmoid* function, $\mathbf{C}$ is a matrix of size $Q/2 \times 2H_0$, and $\mathbf{d}$ is a vector of size $Q/2$. $\mathbf{C}$ and $\mathbf{d}$ are obtained by NN training. Vectors $\tilde{\mathbf{h}}'_k$ and $\tilde{\mathbf{h}}''_k$ are obtained by normalizing vectors $\mathbf{h}'_k$ and $\mathbf{h}''_k$ so that each element of $\tilde{\mathbf{h}}'_k$ and $\tilde{\mathbf{h}}''_k$ has zero mean and unit variance. Vector $\hat{\mathbf{m}}_k$ provides the estimates of the message bits in a corresponding $Q/2$-tuple, that is:

$$\hat{\mathbf{m}}_k = (\hat{m}(kQ/2), \dots, \hat{m}((k+1)Q/2-1)). \quad (31)$$

The Deepcode decoder from [1] is recovered by setting $\gamma_l = 0, l = 0, 1, ..., P$ in *(28)*.

3. TRANSCEIVER TRAINING

The coding and modulation schemes used in conventional communication systems are optimized for a given SNR range. We take the same approach for DNN-based codes: as DNN code training produces different codes depending

on the training SNR, we divide the target range of forward SNRs into (small) non-overlapping intervals and select a single training SNR within each interval.

Encoder and decoder are implemented as two separate DNNs whose coefficients are determined through a joint training procedure. The training procedure consists in the transmission of batches of randomly generated messages. The number of batches is 2×10^4, where each batch contains 2×10^3 messages. DNN coefficients are updated by an Adaptive Moment (ADAM) estimation optimizer based on the *Binary Cross-Entropy* (BCE) loss function. For each batch, a loss value is obtained by computing the BCE between the messages in that batch and the corresponding decoder outputs. The learning rate is initially set to 0.02 and divided by 10 after the first group of 10^3 batches. The gradient magnitude is clipped to 1.

By monitoring the BCE loss value throughout the entire training session, we noticed that the loss trajectory has high peaks which appear more frequently during the initial phases of training. Those peaks indicate that the training process is driving the encoder/decoder NNs away from their optimal performances. In order to mitigate the detrimental effect of the above events, the following countermeasures have been taken:

- usage of a larger batch size, 10 times larger than [1]. Usage of large batches stabilizes training[1] and accelerates convergence of NN weights towards values that produce good performance;
- implementation of a training *roll-back* mechanism that discards the NN weight updates of the last epoch if the loss value produced by the NNs with updated weights is at least 10 times larger than the loss produced by the NNs with previous weights.

As we observed that the outcome of training is sensitive to the random number generators' initialization, each training is repeated three times with different initialization seeds. For each repetition, we record the final NN weights and the NN weights that produced the smallest loss during training. After training, Link-Level Simulations (LLS) are performed using all the recorded weights. The set of weights that provides the lowest Block Error Rate (BLER) is kept and the others are discarded.

As described in Subsection 2.3 and illustrated in Fig. 2, the PSG output is normalized so that each coded symbol has zero mean and unit variance. During NN training, normalization subtracts the *batch mean* from the PSG output and divides the result of subtraction by the batch standard deviation. After training, *encoder calibration* is performed in order to compute the mean and the variance of the RNN outputs over a given number of codewords. Calibration is done over 10^6 codewords in the simulations here reported. In LLS, normalization is done using the mean and variance values computed during calibration.

[1] By *training stabilization* we mean that the loss function produces smoother trajectories during training.

The training strategy for the encoder's codeword and symbol power levels has been optimized empirically. The levels are initialized to unit value, and kept constant for a given number of epochs as early start of training produces codes with poor performance. On the other hand, if training of levels is started too late, they remain close to their initial unit value, and therefore produce no benefits. It has been found empirically that starting to train codeword power levels at epoch 100 and symbol power levels at epoch 200 provides the best results.

As suggested in [1], it may be beneficial to perform training with longer messages compared to link level evaluation as training with short messages does not produce good codes. According to our observations, training with longer messages, twice the length of LLS messages, is beneficial. However, according to our observations, the benefit of using longer messages vanishes when training with larger batches. Therefore, in our evaluations the length of training messages and LLS messages is the same.

The above training method produces codes with better performance compared to the method of [1], as the performance evaluations of Section 4 will show. Training parameters are summarized in Table 3.

4. PERFORMANCE EVALUATIONS

In this section, we assess the BLER performance of DEF codes and compare their performance with the performance of the NR LDPC code reported in [12] and the performance of Deepcode [1] for the same Spectral Efficiency (SE). The SE is defined as the ratio of the number of information bits L over the number of forward-channel time-frequency resources used for transmission of the corresponding codeword. As each time-frequency resource carries a complex symbol, and since each complex symbol is produced by combining two consecutive real symbols, we have

$$SE \triangleq \frac{Q}{1+P} \text{ [bits/s/Hz]}. \qquad (32)$$

The forward-channel and feedback-channel impairments are modeled as AWGN with variance $\sigma_n^2 = 1/SNR$ and $\sigma_{FB}^2 = 1/SNR_{FB}$, respectively. The training forward SNR and LLS forward SNR are the same; the feedback channel is noiseless.

The set of parameters used in the performance evaluations is shown in Table 4. For DEF code performance evaluations, we show that even the shortest feedback extensions – corresponding to the δ and γ parameters of Table 4

Table 3 – Training parameters.

Training parameter	Value
Number of epochs	2000
Number of batches per epoch	10
Number of codewords per batch	2000
Training message length [bits]	50
Starting epoch for codeword-level weights training	100
Starting epoch for symbol-level weights training	200

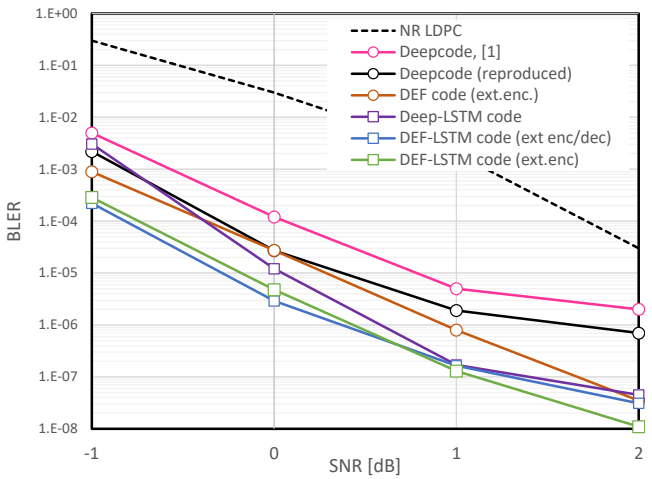

Fig. 4 – Performance comparison of Deepcode, DEF codes, LSTM-based Deepcode, and DEF-LSTM codes. Spectral efficiency is 0.67 bits/s/Hz ($Q = 2,\ P = 2$) .

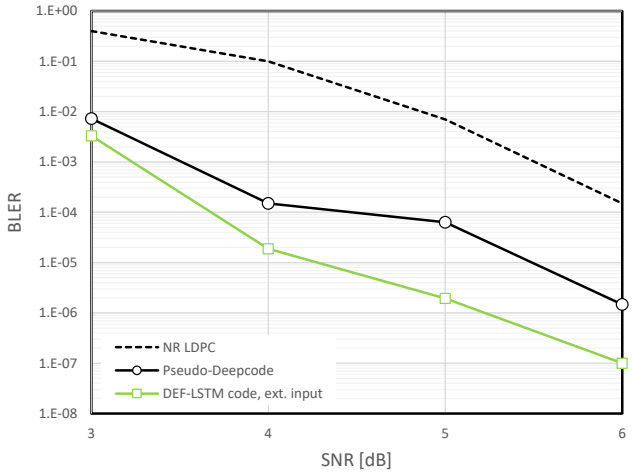

Fig. 5 – Performance comparison of Deepcode, pseudo-Deepcode, and DEF-LSTM code with extended encoder input. Spectral efficiency is 1.33 bits/s/Hz ($Q = 4$, $P = 2$).

– produces significant gains. The investigation of performance with larger feedback extensions is left for future work. Details of the evaluated architectures are reported in Table 5.

Fig. 4 shows the Block Error Rate (BLER) vs. forward SNR of several codes with $SE = 0.67$ bits/s/Hz. The plot shows Deepcode [1] (pink curve), Deepcode obtained by the training method of Section 3 (solid black curve), DEF code with extended encoder input (orange curve), Deepcode with LSTM-based encoder and decoder NNs (purple curve), DEF code with extended encoder input (green curve) and DEF code with extended encoder and decoder input (blue curve). All DNN-based codes use second-order modulation (i.e., $Q = 2$) and $P = 2$ parity symbols per systematic symbol. Thus, the corresponding SE is 0.67 bits/s/Hz. The performance of the NR LDPC code as reported in [12] with the same SE (QPSK modulation, code rate 1/3) is shown by a dashed black curve.

Based on the data shown in Fig. 4, the following observations are made:

- The DEF code with extended encoder input (orange curve) has better performance than Deepcode (solid black curve).
- The DEF-LSTM codes (green and blue curves) have the best performance among all the evaluated codes.
- The DEF-LSTM code with extended encoder input and DEF-LSTM code with extended encoder/decoder input have similar performance except for high SNRs, where the former performs slightly better.
- DEF-LSTM codes (green and blue curve) outperform NR LDPC (dashed black curve) by at least three orders of magnitude BLER for all SNRs.
- The training method of Section 3 (black curve) produces codes with better performance than the training method of [1] (pink curve).

Based on the first observation above, it can be concluded that *encoder* input extension produces performance improvements. Subsequent observations highlight that the encoder input extension provides performance improvements when combined with LSTM. However, based on the observation in the third bullet, we can conclude that *decoder* input extension brings no benefits compared to *encoder* input extension. Moreover, the above performance evaluations show that usage of LSTM in the encoder and decoder provides significant performance improvements compared to RNN/GRU based codes.

Figure 5 shows the BLER performance of DNN-based codes with modulation order $Q = 4$, the corresponding SE is 1.33 bits/s/Hz. As Deepcode [1] is not defined for SEs higher than 0.67 bits/s/Hz,

Table 4 – Evaluation parameters.

DEF code parameter	Selected values
K [symbols]	50
P	2
H_0	50
# zero-padding bits	1
Encoder input extensions	$(\delta_0, \delta_1, \delta_2) = (1, 2, 2)$
Decoder input extensions	$(\gamma_0, \gamma_1, \gamma_2) = (1, 1, 1)$

Table 5 – Evaluated architectures.

Code	Encoder NN (type, #layers)	Decoder NN (type, #layers)
Deepcode	RNN, 1	bidir. GRU, 2
DEF code	RNN, 1	bidir. GRU, 2
Deep-LSTM code	LSTM, 1	bidir. LSTM, 2
DEF-LSTM code	LSTM, 1	bidir. LSTM, 2

we implemented a *pseudo-Deepcode* by replacing the Deepcode modulator with a modulator of order $Q = 4$. Results show that the DEF-LSTM code has better performance compared to the pseudo-Deepcode as its BLER is significantly lower in the whole range of SNR that we evaluated. The DEF-LSTM code BLER gain over pseudo-Deepcode is larger than one order of magnitude for SNR=5 dB and 6 dB. Moreover, the DEF-LSTM code outperforms NR LDPC (dashed black curve) by at least three orders of magnitude BLER for SNR ≥ 4 dB.

5. CONCLUSION AND FURTHER WORK

A new deep neural network-based error correction encoder architecture for channels with feedback has been presented. The new architecture generates parity symbols based on feedbacks in longer time windows compared to prior architectures, thereby introducing long-range dependencies between parity symbols within each codeword.

It has been shown that the codes designed according to the DEF architecture achieve lower error rates than any other code designed for channels with feedback. As long-range dependencies between parity symbols are a necessary ingredient of all good error correction codes, it is expected that further performance improvements can be obtained by increasing the length of the feedback time windows.

Moreover, by a suitable selection of the modulation order, we showed that these codes can adapt to the forward channel quality, thereby providing the maximum spectral efficiency that is attainable for the given forward channel quality.

In this work, DEF codes have been designed and evaluated for forward channels impaired by additive white Gaussian noise and noiseless feedback, where the forward SNR has been assumed to be perfectly known at design time (NN training) and during LLS. Code design with imperfect SNR knowledge and evaluations in more realistic scenarios, such as channels with fading and noisy feedback, are interesting subjects that will need to be addressed in order to make these codes applicable in real transmission systems. However, these topics require further thorough investigation and therefore are left for future works.

REFERENCES

[1] H. Kim, Y. Jiang, S. Kannan, S. Oh, P. Viswanath, "Deepcode: feedback codes via deep learning," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 194-206, May 2020.

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

[3] D. J. C. MacKay, "Good error correcting codes based on very sparse matrices", *IEEE Transactions on Information Theory,* vol. 45, no. 2, pp. 399-431, Mar. 1999.

[4] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes," *International Conference on Communications, ICC'93,*Geneva, Switzerland, pp. 1064-70, May 1993.

[5] S. Lin, D. Costello, *Error Control Coding*, Prentice-Hall, Englewood Cliffs, NJ: 1983, 2011.

[6] Y. Polyanskiy, H. V. Poor and S. Verdú, "Feedback in non-asymptotic regime," *IEEE Transactions on Information Theory,*, vol. 57, no. 8, pp. 4903-4925, Aug. 2011.

[7] J. Schalkwijk, "A coding scheme for additive noise channels with feedback-I: No bandwidth constraint," *IEEE Transactions on Information Theory*, vol. 12, no. 2, pp. 172-182, Aug. 1966.

[8] M. Horstein, "Sequential transmission using noiseless feedback," *IEEE Transactions on Information Theory*, vol. 9, no. 3, pp. 136-143, July 1963.

[9] J. M. Ooi and G. W. Wornell, "Fast iterative coding techniques for feedback channels," *IEEE Transactions on Information Theory,*, vol. 44, no. 7, pp. 2960-2976, Nov. 1998.

[10] Z. Ahmad, Z. Chance, D. J. Love and C. Wang, "Concatenated coding using linear schemes for Gaussian broadcast channels with noisy channel output feedback," *IEEE Transactions on Communications*, vol. 63, no. 11, pp. 4576-4590, Nov. 2015.

[11] K. Vakilinia, S. V. S. Sudarsan, V. S. Ranganathan, D. Divsalar and R. D. Wesel, "Optimizing transmission lengths for limited feedback with non-binary LDPC examples," *IEEE Transaction on communications,* vol. 64, no. 6, pp. 2245-2257, June 2016.

[12] Huawei, HiSilicon, "Performance evaluation of LDPC codes for NR eMBB data," R1-1713740, 3GPP RAN1 meeting #90, Prague, Czech Republic, August 21–25, 2017.

AUTHORS

Anahid Robert Safavi received her Ph.D. degree in signal processing from Télécom Paris, France in 2003. Since then, she has worked as a researcher in the telecommunications industry with Motorola Mobile device, Motorola Labs and, over the past twelve years, with Huawei Technologies. From 2004 to 2008 with Motorola she was involved in receiver design for GSM and IEEE specification. More recently, from 2008 to 2020, she contributed to the preparation and specification of 3GPP. Since 2021, she has been working on 5G receiver design. She holds more than 20 patents and is author of several IEEE conference and journal papers. Her research interests include deep learning applied to air interface, mMIMO, MIMO, iterative receivers, channel coding and multiple access.

Alberto G. Perotti received his Ph.D. degree in telecommunications from Politecnico di Torino, Italy, in 2003. He is a principal research engineer at Huawei Technologies, where he is involved in wireless networks' physical layer research and standardization. Prior to joining Huawei, he held research and teaching positions at Politecnico di Torino, and has been head of networks and wireless communications research at CSP-ICT innovation, Turin, Italy. His research interests cover channel coding and modulation, multiple access, and software-defined radios. He serves as IEEE Communications Magazine technical editor and lead editor for the *Mobile Communications and Networks* series.

Branislav M. Popović received his Ph.D. degree in electrical engineering from the School of Electrical Engineering, University of Belgrade, Serbia. He is a Huawei Fellow. Prior to joining Huawei Technologies, Stockholm, Sweden, in 2001, he was with Marconi, Stockholm, from 2000 to 2001, Ericsson, Stockholm, from 1994 to 2000, and the Institute of Microwave Techniques and Electronics, Belgrade, from 1984 to 1994.

Mahdi Boloursaz Mashhadi received his Ph.D., M.Sc., and B.Sc. degrees in electrical engineering from Sharif University of Technology (SUT), Tehran, Iran, in 2018, 2013 and 2011, respectively. He is currently a post-doctoral research associate at the Intelligent Systems and Networks (ISN) research group, Imperial College London, UK. He also worked as a research associate at the University of Central Florida (UCF), and Queens' University, Canada, and as a lecturer at Sharif University of Technology (SUT), Tehran, Iran. His research interests include the areas of wireless communications, machine learning, and sparse/statistical signal processing. He serves as a reviewer for IEEE Transactions on Wireless Communications, IEEE Transactions on Communications, and IEEE Transactions on Signal Processing.

Deniz Gündüz received his Ph.D. degree in electrical engineering from NYU Tandon School of Engineering (formerly Polytechnic University) in 2007. He is currently a professor of information processing in the Electrical and Electronic Engineering Department of Imperial College London, UK, and a part-time faculty member at the University of Modena and Reggio Emilia, Italy. Previously he served as a postdoctoral research associate at Princeton University, as a consulting assistant professor at Stanford University, and as a research associate at CTTC in Spain. He has held visiting positions at University of Padova (2018-2020) and Princeton University (2009-2012). Prof. Gündüz is a Distinguished Lecturer for the IEEE Information Theory Society (2020-22). He is the recipient of the IEEE Communications Society - Communication Theory Technical Committee (CTTC) Early Achievement Award in 2017, a Starting Grant of the European Research Council (ERC) in 2016, and several best paper awards. He is an area editor for the IEEE Transactions on Communications, the IEEE Transactions on Information Theory, and the IEEE Journal on Selected Areas in Communications (JSAC) Special Series on Machine Learning in Communications and Networking. He also serves as an editor of the IEEE Transactions on Wireless Communications. His research interests lie in the areas of communications and information theory, machine learning, and privacy.