Deepcode: Feedback Codes via Deep Learning

Hyeji Kim, Yihan Jiang, Sreeram Kannan, Sewoong Oh, Pramod Viswanath

***Abstract*—The design of codes for communicating reliably over a statistically well defined channel is an important endeavor involving deep mathematical research and wide-ranging practical applications. In this work, we present the first family of codes obtained via deep learning, which significantly outperforms state-of-the-art codes designed over several decades of research. The communication channel under consideration is the Gaussian noise channel with feedback, whose study was initiated by Shannon; feedback is known theoretically to improve reliability of communication, but no practical codes that do so have ever been successfully constructed.**

We break this logjam by integrating information theoretic insights harmoniously with recurrent-neural-network based encoders and decoders to create novel codes that outperform known codes by 3 orders of magnitude in reliability and achieve a 3dB gain in terms of SNR. We also demonstrate several desirable properties of the codes: (a) generalization to larger block lengths, (b) composability with known codes, and (c) adaptation to practical constraints. This result also has broader ramifications for coding theory: even when the channel has a clear mathematical model, deep learning methodologies, when combined with channel-specific information-theoretic insights, can potentially beat state-of-the-art codes constructed over decades of mathematical research.

***Index Terms*—Channel coding, Deep learning, Neural networks, Recurrent neural networks, Feedback communication, Schalkwijk–Kailath scheme**

I. Introduction

The ubiquitous digital communication enabled via wireless (e.g. WiFi, mobile, satellite) and wired (e.g. ethernet, storage media, computer buses) media has been the workhorses underlying the current information age. The advances of reliable and efficient digital communication have been primarily driven by the design of codes which allow the receiver to recover messages reliably and efficiently under noisy conditions. The discipline of coding theory has made significant progress in the past seven decades since Shannon's celebrated work in 1948 [1]. As a result, we now have near optimal codes in a canonical setting, namely, additive white Gaussian noise (AWGN) channel. However, several channel models of great practical interest lack efficient and practical coding schemes.

A channel with *feedback* (from the receiver to the transmitter) is an example of a long-standing open problem with significant practical importance. Modern wireless communication includes feedback in one form or the other; for example, the feedback can be the received value itself, or quantization of the received value or an automatic repeat request (ARQ) [2]. Accordingly, there are different models for channels with feedback, and among them, the AWGN channel with *output* feedback is a model that captures the essence of channels with feedback; this model is also classical, introduced by Shannon in 1956 [3]. In this channel model, the received value is fed back (with unit time delay) to the transmitter without any processing (refer to Figure 1). Designing codes for this channel via deep learning approaches is the central focus of this paper.

Whereas the output feedback does not improve the Shannon capacity of the AWGN channel [3], it is known to provide better reliability at finite block lengths [4]. On the other hand, practical coding schemes have not been successful in harnessing the feedback gain thereby significantly limiting the use of output feedback in practice. This state of the art is at odds with the theoretical predictions of the gains in reliability via using feedback: the seminal work of Schalkwijk and Kailath [4] proposed a (theoretically) achievable scheme (S-K scheme) with superior reliability guarantees. However, the S-K scheme is shown to be extremely sensitive to both the precision of the numerical computation and noise in the feedback [5], [6]. Whereas several works extended the S-K scheme to noisy feedback settings [7], [8], [9], the success has been limited. For example, the scheme of [8] is designed for channels with noisy feedback, but not only is the reliability poor, it is often independent of the feedback quality as shown in Figure 12, suggesting that the feedback data is not being fully exploited. More generally, it has been proven that no *linear* code incorporating the noisy output feedback can achieve a positive rate of communications [9]. This is especially troubling since all practical codes are linear and linear codes are known to achieve capacity (without feedback) [10], whereas [9] proposes an *asymptotically* optimal and nonlinear coding scheme for channels with noisy feedback based on a

H. Kim is with the Samsung AI Research in Cambridge, United Kingdom. Y. Jiang and S. Kannan are with the Department of Electrical Engineering at University of Washington. S. Oh is with the Department of Computer Science and Engineering at University of Washington. P. Viswanath is with the Department of Electrical Engineering at University of Illinois at Urbana Champaign. Email: hkim1505@gmail.com, yihanrogerjiang@gmail.com, ksreeram@uw.edu, sewoong@cs.washington.edu, pramodv@illinois.edu

This paper is an extended version of work appeared in the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018).

three-phase detection and retransmission protocol.

Throughout the paper, we consider a finite-length block coding with a fixed rate setting (i.e., the total number of channel usages does not change). Under the variable length setting (i.e., total number of channel usages varies and depends on the feedback) with noisy output feedback, the transmitter and the receiver may not be in an agreement on whether the transmission is over or not, and this can cause an error on current and the upcoming transmissions [7]. Under the variable length setting, [11], [12] proposes a coding scheme for channels with noisy feedback and demonstrates that the improvement in reliability function resulting from the variable-length coding is not fragile to the noise in the feedback. In [13], the authors propose a variable-length code with *active* feedback (i.e., coded feedback as opposed to the output feedback) and show that their code improves the reliability under noisy feedback if the feedback Signal-to-Noise Ratio (SNR) is sufficiently larger than the forward SNR.

In this paper, we demonstrate new neural network-driven encoders (with matching decoders) that operate significantly better (100–1000 times in Bit Error Rate (BER) and 3dB gain in SNR) than state of the art on the AWGN channel with (noisy) output feedback. We show that architectural insights from simple communication channels with feedback, when coupled with recurrent neural network architectures, can discover novel codes. We consider Recurrent Neural Network (RNN) parameterized encoders (and decoders), which are inherently *nonlinear* and map information bits *directly* to real-valued transmissions in a sequential manner.

Designing codes driven by deep learning has been of significant interest recently, starting from [14] which proposes an autoencoder framework for communications. In [14], it is demonstrated that for classical AWGN channels, feedforward neural codes can mimic the performance of a well-known code for a short block length (4 information bits). Extending this idea to orthogonal frequency division multiplex (OFDM), [15], [16] show that neural codes can mimic the performance of state-of-the-art codes for short block lengths (8 information bits). Several results extend the autoencoder idea to other settings of AWGN channels [17] and modulation [18]. Beyond AWGN channels, [19] considers the problem of communicating a complicated source (text) over erasure channels and shows that RNN-based neural codes which map raw texts directly to codewords can beat the state-of-the art codes, when the reliability is evaluated by human perception (as opposed to bit error rate). Deep learning has been applied also in the problem of designing decoders for existing encoders [20], [21], [22], [23], [24], [25], [26], demonstrating the efficiency, robustness, and adaptivity of neural decoders over the existing decoders. In a different context, for distributed computation, where an encoder adds redundant computations so that the decoder can reliably approximate the desired computations under unavailabilities, [27] showed that neural network based codes can beat the state of the art codes.

While several works in the past years apply deep learning for channel coding, very few of them consider the design of novel codes using deep learning (rather than decoders). Furthermore, none of them are able to beat state-of-the-art channel codes on a canonical (well known) channel in terms of the standard reliability metric. We demonstrate first family of codes obtained via deep learning which outperforms state-of-the-art codes, signaling a potential shift in code design, which historically has been driven by individual human ingenuity with sporadic progress over the decades. Henceforth, we call this new family of codes *Deepcode*. We also demonstrate the superior performance of variants of Deepcode under a variety of practical constraints. Our main contributions are as follows:

1) We demonstrate Deepcode – a new family of RNN-driven neural codes that has *three orders of magnitude* better reliability than state of the art with both noiseless and noisy feedback (and 3dB gain in SNR). Our results are significantly driven by the intuition obtained from information and coding theory, in designing a series of progressive improvements in the neural network architectures. We provide a detailed comparison on the complexity; while Deepcode has complexity linear in block length, without any optimization of complexity, Deepcode is more complex than traditional codes (Section III and IV).
2) We show that variants of Deepcode significantly outperform state-of-the art codes under a variety of practical constraints (example: delayed feedback, very noisy feedback link) (Section IV).
3) We show *composability*: Deepcode naturally concatenates with a traditional inner code and demonstrates continued improvements in reliability as the block length increases (Section IV).
4) Our interpretation and analysis of Deepcode provide a guidance on the fundamantal understanding of how the feedback can be used and some information theoretic insights into designing codes for channels with feedback (Section V).
5) We discuss design decisions and demonstrate practical gains of Deepcode in practical cellular communication systems (Section VI).

II. Problem formulation

The most canonical channel studied in the literature (example: textbook material [28]) and also used in modeling practical scenarios (example: 5G LTE standards) is the

Additive White Gaussian Noise (AWGN) channel *without* feedback. Concretely, the encoder takes in K information bits jointly, $\mathbf{b} = (b_1, \cdots, b_K) \in \{0,1\}^K$, and outputs n real valued signals to be transmitted over a noisy channel (sequentially). At the i-th transmission for each $i \in \{1, \ldots, n\}$, a transmitted symbol $x_i \in \mathbb{R}$ is corrupted by an independent Gaussian noise $n_i \sim \mathcal{N}(0, \sigma^2)$, and the decoder receives $y_i = x_i + n_i \in \mathbb{R}$. After receiving the n received symbols, the decoder makes a decision on which information bit sequence $\mathbf{b}$ was sent, out of 2^K possible choices. The goal is to maximize the probability of correctly decoding the received symbols and recover $\mathbf{b}$.

Both the encoder and the decoder are functions, mapping $\mathbf{b} \in \{0,1\}^K$ to $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^n$ to $\hat{\mathbf{b}} \in \{0,1\}^K$, respectively. The design of a good code (an encoder and a corresponding decoder) addresses both (i) the statistical challenge of achieving a small error rate; and (ii) the computational challenge of achieving the desired error rate with efficient encoder and decoder. Almost a century of progress in this domain of coding theory has produced several innovative codes that efficiently achieve small error rate, including convolutional codes, Turbo codes, LDPC codes, and polar codes. These codes are known to perform close to the fundamental limits of reliable communication [29].

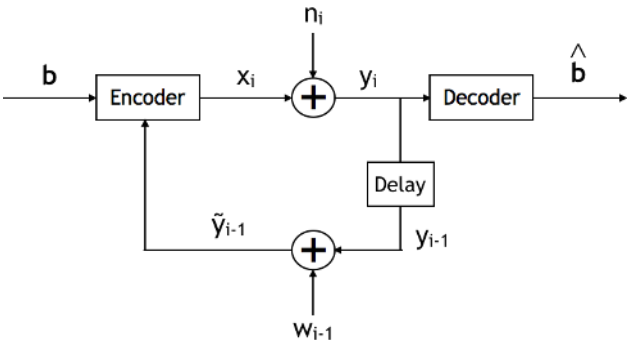


Fig. 1: AWGN channel with noisy output feedback

In a canonical AWGN channel *with* noisy feedback, the received symbol y_i is transmitted back to the encoder after one unit time of delay and via another additive white Gaussian noise *feedback channel* (Figure 1). The encoder can use this feedback symbol to sequentially and adaptively decide what symbol to transmit next. At time i the encoder receives a noisy view of what was received at the receiver (in the past by one unit time), $\tilde{y}_{i-1} = y_{i-1} + w_{i-1} \in \mathbb{R}$, where the noise is independent and distributed as $w_{i-1} \sim \mathcal{N}(0, \sigma_F^2)$. Formally, an *encoder* is now a function that sequentially maps the information bit vector $\mathbf{b}$ and the feedback symbols $\tilde{y}_1^{i-1} = (\tilde{y}_1, \cdots, \tilde{y}_{i-1})$ received thus far to a transmit symbol x_i: $f_i : \quad (\mathbf{b}, \tilde{y}_1^{i-1}) \mapsto x_i, \qquad i \in \{1, \cdots, n\}$ and a *decoder* is a function that maps the received sequence $y_1^n = (y_1, \cdots, y_n)$ into estimated information bits: $g : \quad y_1^n \mapsto \hat{\mathbf{b}} \in \{0,1\}^K$.

The standard measures of performance are the average bit error rate (BER) defined as $\mathrm{BER} \equiv (1/K) \sum_{i=1}^K \mathbb{P}(b_i \neq \hat{b}_i)$ and the block error rate (BLER) defined as $\mathrm{BLER} \equiv \mathbb{P}(\mathbf{b} \neq \hat{\mathbf{b}})$, where the randomness comes from the forward and feedback channels and any other sources of randomness that might be used in the encoding and decoding processes. It is standard (both theoretically and practically) to have an average power constraint, i.e., $(1/n)\mathbb{E}[\|\mathbf{x}\|^2] \leq 1$, where $\mathbf{x} = (x_1, \cdots, x_n)$ and the expectation is over the randomness in choosing the information bits $\mathbf{b}$ uniformly at random, the randomness in the noisy feedback symbols $(\tilde{y}_1, \cdots, \tilde{y}_n)$ and any other randomness used in the encoder.

A. Results preview

While the capacity of the channel remains the same in the presence of feedback [3], the reliability can increase significantly as demonstrated by the celebrated result of Schalkwijk and Kailath (S-K) [4]. Although the optimal theoretical performance is met by the S-K scheme, critical drawbacks make it fragile. Theoretically, the scheme critically relies on exactly noiseless feedback (i.e. $\sigma_F^2 = 0$), and does not extend to channels with even arbitrarily small amount of noise in the feedback (i.e. $\sigma_F^2 > 0$). The scheme is also very sensitive to numerical precisions; we see this in Figure 2, where the numerical errors dominate the performance of the S-K scheme, with a practical choice of MATLAB implementation with a precision of 64 bits to represent floating-point numbers.

Even with a noiseless feedback channel with $\sigma_F^2 = 0$, which the S-K scheme is designed for, it is outperformed significantly by our proposed Deepcode (described in detail in Section III). At moderate SNR of 2 dB, Deepcode can outperform S-K scheme by three orders of magnitude in BER. In Figure 2 (top), the resulting BER is shown as a function of the Signal-to-Noise Ratio (SNR) defined as $-10 \log_{10} \sigma^2$, where we consider the setting of rate $1/3$ and information block length of $K = 50$ (hence, $n = 150$). Also shown as a baseline is an LTE turbo code which does not use any feedback. Deepcode exploits the feedback symbols to achieve a significant gain of two orders of magnitude consistently over the Turbo code for all SNR. In Figure 2 (bottom), BLER of Deepcode is shown as a function of the Signal-to-Noise Ratio (SNR), together with state-of-the art polar, LDPC, and convolutional codes in a 3GPP document for 5G [30] (we refer to Appendix VIII for the details of these codes). Deepcode significantly improves over all state-of-the-art codes of the similar block-length and the same rate. Also plotted as a baseline are the theoretically estimated performance of the best code with no efficient

decoding schemes. This impractical baseline lies between approximate achievable BLER (labelled Normapx in the figure) and a converse to the BLER (labelled Converse in the figure) from [29], [31]. We note that there are schemes proposed more recently that address the sensitivity to noise in the output feedback, a major drawback of the S-K scheme (e.g., [32] and [8]). However, these schemes either still suffer from similar sensitivity to numerical precisions at the decoder due to the uniform message constellation as in the S-K scheme [32], or are often incapable of exploiting the feedback information [8] as we illustrate in Figure 12 in experiments with noisy feedback.

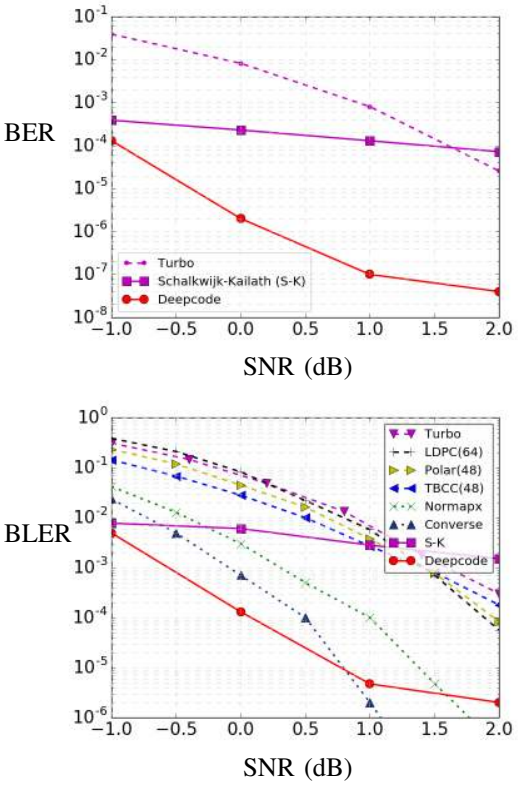


Fig. 2: Deepcode significantly outperforms the baseline of S-K and Turbo code when information block length is 50 and noiseless feedback is available in BER (top) and BLER (bottom). Deepcode also outperforms all state-of-the art codes (without feedback) in BLER (bottom).

III. DEEPCODE: NEURAL ENCODER AND DECODER

A natural strategy to create a feedback code is to utilize a recurrent neural network (RNN) as an encoder since (i) communication with feedback is naturally a sequential process and (ii) we can exploit the sequential structure for an efficient decoding. We propose representing the encoder and the decoder as RNNs, training them jointly under AWGN channels with noisy feedback, and minimizing the error in decoding the information bits. However, in our experiments, we find that this strategy by itself is insufficient to achieve any performance improvement with feedback; there are several design elements that need to be carefully chosen in constructing and training a RNN based code.

We exploit information theoretic insights to enable improved performance, by considering the coding scheme for erasure channels with feedback: here transmitted bits are either received perfectly or erased, and whether the previous bit was erased or received perfectly is fed back to the transmitter. In such a channel, the following two-phase scheme can be used: transmit a block of symbols, and then transmit whichever symbols were erased in the first block (and ad infinitum). This motivates a two-phase scheme, where uncoded bits are sent in the first phase, and then based on the feedback in the first phase, coded bits are sent in the second phase; thus the code only needs to be designed for the second phase. We show in this section that these intuitions can be critically employed to innovate neural network architectures for coding on AWGN channels with feedback. Even within this two-phase paradigm, several architectural choices need to be made. In the following, we show the baseline neural network architectures (Scheme A) and a series of improvements (Schemes B,C,D) made based on the typical error analysis.

Our experiments focus on the setting of rate 1/3 and information block length of 50 for concreteness[1]. That is, the encoder maps $K = 50$ message bits to a codeword of length $n = 150$. We discuss generalizations to longer block lengths in Section IV.

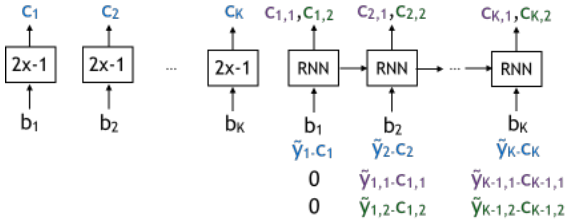
Scheme A. RNN based feedback encoder/decoder (RNN (linear) and RNN (tanh)).

We propose a baseline encoding scheme that progresses in two phases. In the first phase, the K raw information bits are sent (uncoded) over the AWGN channel. In the second phase, $2K$ coded bits are generated based on the information bits $\mathbf{b}$ and (delayed) output feedback and sequentially transmitted (so that the total rate is fixed as 1/3).

Encoding. The architecture of the encoder is shown in Figure 3. The architectures for RNN (tanh) and RNN (linear) feedback codes are equivalent except the activation function in RNN; RNN (tanh) encoder uses a tanh activation while RNN (linear) encoder uses a linear activation (for both the recurrent and output activation). In the first phase of the encoding process, the encoder simply transmits the K raw message bits via binary

[1]Source codes are available under https://github.com/hyejikim1/feedback_code (Keras) and https://github.com/yihanjiang/feedback_code (PyTorch)

phase shift keying (BPSK). That is, the encoder maps b_k to $c_k = 2b_k - 1$ for $k \in \{1, \cdots, K\}$, and stores the feedback $\tilde{y}_1, \cdots, \tilde{y}_K$ for later use. In the second phase, the encoder generates a coded sequence of length $2K$ (length $(1/r - 1)K$ for general rate r code) through a single directional RNN. In particular, each k-th RNN cell generates two coded bits $c_{k,1}, c_{k,2}$ for $k \in \{1, \ldots, K\}$, which uses both the information bits and (delayed) output feedback from the earlier raw information bit transmissions. The input to the k-th RNN cell is of size four: b_k, $\tilde{y}_k - c_k$ (the estimated noise added to the k-th message bit in phase 1) and the most recent two noisy feedbacks from phase 2: $\tilde{y}_{k-1,1} - c_{k-1,1}$ and $\tilde{y}_{k-1,2} - c_{k-1,2}$. Note that we use $\tilde{y}_{k,j} = c_{k,j} + n_{k,j} + w_{k,j}$ to denote the feedback received from the transmission of $c_{k,j}$ for $k \in \{1, \cdots, K\}$ and $j \in \{1, 2\}$, and $n_{k,j}$ and $w_{k,j}$ are corresponding forward and feedback channel noises, respectively.



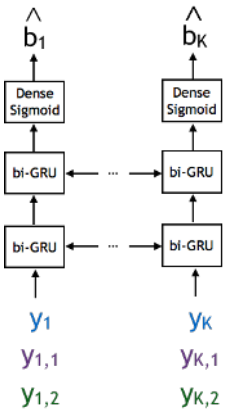
Layer	Output dimension
Input	(K, 4)
RNN (linear or tanh)	(K, 50)
Dense (sigmoid)	(K, 2)
Normalization	(K, 2)

Fig. 3: RNN encoder for Scheme A.

To generate codewords that satisfy the power constraint, we put a normalization layer to the RNN outputs so that each coded bit has a mean 0 and a variance 1. During training, the normalization layer subtracts the batch mean from the output of RNN and divide by the standard deviation of the batch. After training, we compute the mean and the variance of the RNN outputs over 10^6 examples. In testing, we use the precomputed means and variances.

Decoding. We propose a decoding scheme using two layers of bidirectional Gated Recurrent Units (GRU). The architecture of the decoder is shown in Figure 4. Based on the received sequence $\mathbf{y} = (y_1, \cdots, y_k, y_{1,1}, y_{1,2}, y_{2,1}, y_{2,2}, \cdots, y_{K,1}, y_{K,2})$ of length $3K$, the decoder estimates K information bits. For the decoder, we use a two-layered bidirectional Gated Recurrent Unit (GRU), where the input to the k-th GRU cell is a tuple of three received symbols, $(y_k, y_{k,1}, y_{k,2})$.

Training. As illustrated in Figure 5, both the encoder



Layer	Output dim.
Input	(K, 3)
bi-GRU	(K, 100)
Batch Norm.	(K, 100)
bi-GRU	(K, 100)
Batch Norm.	(K, 100)
Dense (sigmoid)	(K, 1)

Fig. 4: RNN decoder for Scheme A.

and decoder are trained *jointly* as in the autoencoder training. We model the whole communication system including the encoder and the channels and the decoder as a large neural network, where the input is a random sequence of message bits and the output is the estimate of the message bit sequence. (We refer to Figure 18 in Appendix IX for a detailed illustration of the encoder and the decoder.)

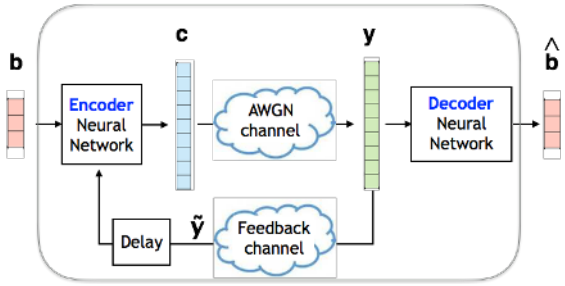


Fig. 5: Autoencoder framework for the joint training of encoder–decoder (illustrated for information block length 3, rate 1/3)

For training examples, we generate a random message bit sequence $\mathbf{b} = (b_1, \cdots, b_K)$ and a random noise sequence for the forward channel (and a random noise sequence for the feedback channel if we consider an AWGN feedback channel). We train the encoder and the decoder jointly via backpropagation through time (on the entire input sequence), where the goal of training is to minimize the binary cross-entropy loss function $\mathcal{L}(\mathbf{b}, \hat{\mathbf{b}}) = \sum_{i=1}^{K} (-b_i \log \hat{b}_i - (1-b_i) \log(1-\hat{b}_i))$. We do the backpropagation through time over a 4×10^6 examples via an Adam optimizer (β_1=0.9, β_2=0.999). We fix the batch size as 200. We randomly initialize weights of the encoder and the decoder. We observe that training with a random initialization of the encoder and the decoder gives a better code compared to initializing with a pre-trained

encoder/decoder by sequential channel codes for non-feedback AWGN channels (e.g. convolutional codes). We also use a decaying learning rate and gradient clipping; we reduce the learning rate by 10 times after training with 10^6 examples, starting from 0.02. Gradients are clipped to 1 if L_2 norm of the gradient exceeds 1 so that we prevent the gradients from getting too large. We do not use any dropout. We find that the choices of training examples are important. Empirically we find that if the length of the training input sequence is too small (e.g., 50), we cannot learn a good structured code. As we use the RNN based encoder and decoder, the learned code generalizes to arbitrary block lengths (e.g., as opposed to a feedforward neural network which only applies to a fixed input length). We set the length of the training input sequence to 100 (and test with input sequence length 50). In generating two sets of noise sequence for the AWGN channels used during the training, we find that it works best to set the SNRs equal to the SNRs to be used in testing. For example, if we would like to learn a code to be used under the 1dB forward channel with 1dB feedback channel, it is best to train with noise sequences generated under those SNRs.

Result. When jointly trained, as shown in Figure 6, a *linear* RNN encoder (- -) achieves performance close to Turbo code that does not use the feedback information at all (-▽-). (To generate plots in Figure 6, we take an average bit error rate over 10^8 bits for SNR = $-1, 0$dB and 10^9 bits for SNR= $1, 2$dB.) With a *non-linear* activation function of $\tanh(\cdot)$, the performance improves, achieving BER close to the existing S-K scheme. Such a gain of non-linear codes over linear ones is in-line with theory [32]. In order to further improve the reliability, we perform typical error analysis and propose modifications to the RNN encoder and decoder architectures (Schemes B,C,D). The improved reliabilities of modified architectures are also shown in Figure 6.

Typical error analysis. Due to the recurrent structure in generating coded bits $(c_{k,1}, c_{k,2})$, the coded bit stream carries more information on the first few bits than last few bits (e.g. b_1 than b_K). This results in more errors in the last information bits, as shown in Figure 7, where we plot the average BER of b_k for $k = \{1, \cdots, K\}$. In the following, we propose modifications to resolve this issue.

Scheme B. RNN feedback code with zero padding (RNN (tanh) + ZP).

In order to reduce high errors in the last information bits, as shown in Figure 7, we apply the zero padding (ZP) technique; we pad a zero in the end of information bits, and transmit a codeword for the padded information bits.

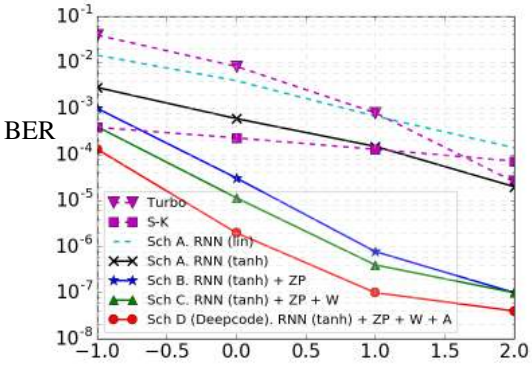


Fig. 6: Building upon a simple linear RNN encoder (Figure 3), we progressively improve the architecture. Eventually with RNN(tanh)+ZP+W+A architecture formally described in Section III, we significantly outperform the baseline of S-K scheme and Turbo code, by several orders of magnitude in the bit error rate, when information block length is 50 and noiseless feedback is available ($\sigma_F^2 = 0$ and forward channel is AWGN).

Encoding and decoding. The encoder and decoder structures with zero padding are shown in Figure 8 and Figure 9, respectively. We maintain the encoder and decoder architecture same as Scheme A (RNN (tanh)) and simply replace the input information bits by information bits padded by a zero; hence, we use $K + 1$ RNN cells in Phase 2 instead of K.

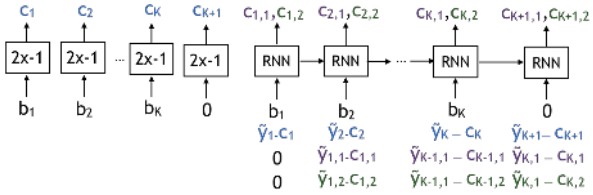


Fig. 8: Encoder for Scheme B.

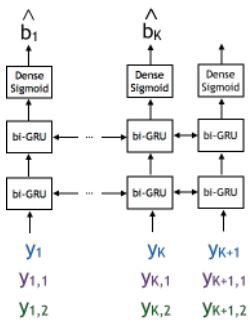


Fig. 9: Decoder for Schemes B,C,D.

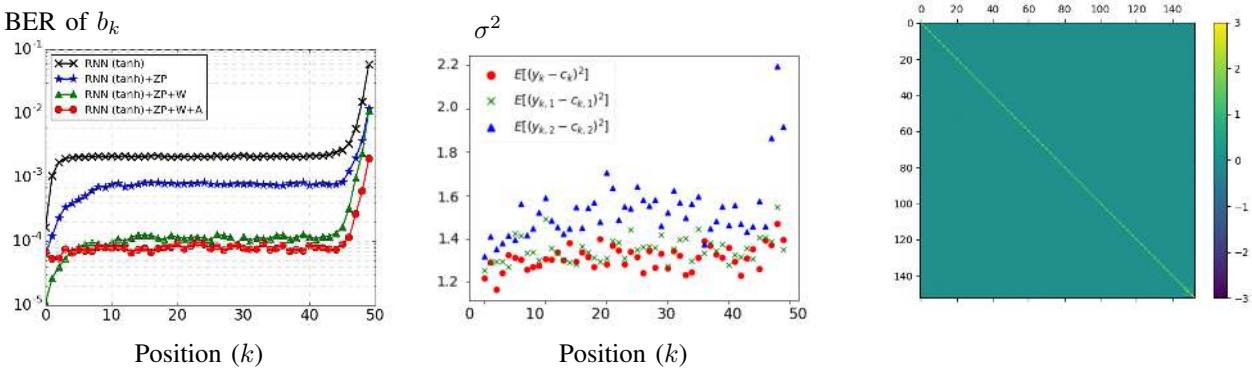


Fig. 7: (Left) A naive RNN(tanh) code gives a high BER in the last few information bits. With the idea of zero padding and power allocation, the RNN(tanh)+ZP+W+A architecture gives a BER that varies less across the bit position, and overall BER is significantly improved over the naive RNN(tanh) code. (Middle) Noise variances across bit position which result in a block error: high noise variance on the second parity bit stream $(c_{1,2}, \cdots, c_{K,2})$ causes a block error. (Right) Noise covariance: Noise sequence which results in a block error does not have a significant correlation across position.

Training. As in training Scheme A, we use back-propagation with binary cross entropy loss. We measure binary crossentropy loss on the information bits of length K only (i.e., ignore the loss on the last bit which corresponds to a zero padding).

Result. By applying zero padding, the BER of the last information bits, as well as other bits, drops significantly, as shown in Figure 7. Zero padding requires a few extra channel usages (e.g. with one zero padding, we map 50 information bits to a codeword of length 153. Actual transmission requires 152 channel usages because the padded zero itself does not need to be transmitted.). However, due to the significant improvement in BER, it is widely used in sequential codes (e.g. convolutional codes and turbo codes).

Typical error analysis. To see if there is a pattern in the noise sequence which makes the decoder fail, we simulate instances of the code and the channel (noise sequence) and collect the corresponding decoding results (whether each instance is decoded correct or wrong). We then look at the first and second order noise statistics which result in the decoding error. In Figure 7 (Middle), we plot the average variance of noise added to b_k in first phase and $c_{k,1}$ and $c_{k,2}$ in the second phase, as a function of k, which results in the (block) error in decoding. From the figure, we make two observations; (i) large noise in the last bits causes an error, and (ii) large noise in $c_{k,2}$ is likely to cause an error, which implies that the raw bit stream and the coded bit streams are not equally robust to the noise – an observation that will be exploited next. In Figure 7 (Right), we plot noise covariances that result in a decoding error. From Figure 7 (Right), we see that there is no particular correlation within the noise sequence that makes the decoder fail.

Scheme C. RNN feedback code with power allocation (RNN(tanh) + ZP + W).

Based on the observation that the raw bit c_k and coded bits $c_{k,1}, c_{k,2}$ are not equally robust, as shown in Figure 7 (Middle), we introduce trainable weights which allow allocating different amount of power to the raw bit stream and coded bit streams.

Encoding and decoding. The encoder and decoder architectures for scheme C are shown in Figure 10 and Figure 9, respectively. Specifically, we introduce three trainable weights (w_0, w_1, w_2) which represent the power allocated to $c_k, c_{k,1}, c_{k,2}$ for all $k \in \{1, \cdots, K\}$, respectively. The weights (w_0, w_1, w_2) satisfies $w_0^2 + w_1^2 + w_2^2 = 3$ so that the average power is preserved (c.f. in Encoder B, we let $\mathbb{E}[c_k^2] = \mathbb{E}[c_{k,1}^2] = \mathbb{E}[c_{k,2}^2] = 1$ and $w_1 = w_2 = w_3 = 1$).

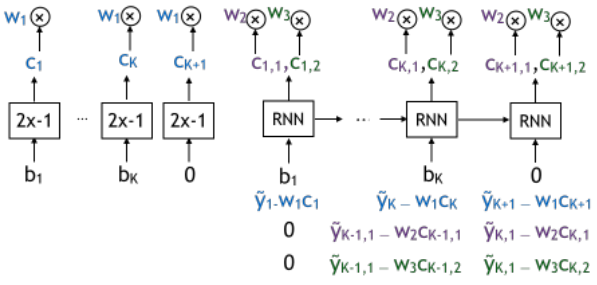


Fig. 10: Encoder for Scheme C.

Training. We initialize w_is by 1 and train the encoder and decoder jointly as we trained Schemes A and B. The trained weights are $(w_1, w_2, w_3) = (1.13, 0.90, 0.96)$ (trained at -1dB). This implies that the encoder uses more power in Phase I, to transmit (raw) information bits. In Phase II, the encoder uses more power on the second parity bits than in the first parity bits.

Result and typical error analysis. By introducing and training these weights, we achieve the improvement in BER as shown in Figures 6 and 7. While the average BER is improved by about an order of magnitude for most bit positions as shown in Figure 7 (Left), the BER of the last bit remains about the same. On the other hand, the BER of first few bits are now smaller, suggesting the following bit-specific power allocation method.

Scheme D. Deepcode: RNN feedback code with bit power allocation (RNN(tanh) + ZP + W + A).

One way to resolve the unbalanced error according to bit position is to use power allocation. Ideally, we would like to reduce the power for the first information bits and increase the power for the last information bits so that we help transmission of last few information bits more than first information bits. However, it is not clear how much power to allow for the first few information bits and the last few information bits. Hence, we introduce a weight vector allowing the power of bits in different position to be different.

Encoding and decoding. The encoder and decoder architectures for scheme D are shown in Figure 11 and Figure 9, respectively. We introduce trainable weights $a_1, a_2, \cdots, a_K, a_{K+1}$ for power allocation in each transmission. To the full generality, we can train all these $K+1$ weights. However, we let $a_5, \cdots, a_{K-4} = 1$ and only train first 4 weights and the last 5 weights, a_1, a_2, a_3, a_4 and $a_{K-3}, a_{K-2}, a_{K-1}, a_K, a_{K+1}$, for two reasons. Firstly, this way we can generalize the encoder to longer block lengths by maintaining the weights for first four and last five weights and fixing the rest of weights as 1s, no matter how many rest weights we have. For example, if we test our code for length 1000 information bits, we can let $a_5, \cdots, c_{996} = 1$. Secondly, the BERs of middle bits do not depend much on the bit position; hence, power control is not as much needed as the first and last few bits.

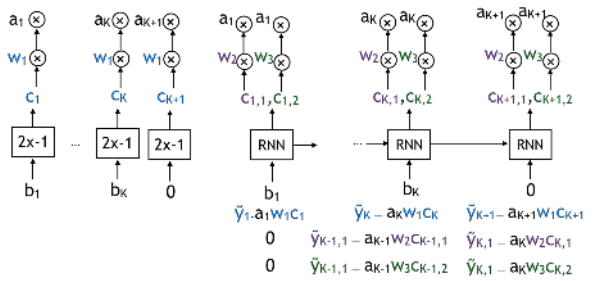


Fig. 11: Encoder for Scheme D: Deepcode.

Training. In training scheme D, we initialize the encoder and decoder as the ones in Scheme C, and then additionally train the weight vectors **a** on top of the trained model, while allowing all weights in the encoder and decoder to change as well. After training, we see that the trained weights are $(a_1, a_2, a_3, a_4) = (0.87, 0.93, 0.96, 0.98)$ and $(a_{K-3}, a_{K-2}, a_{K-1}, a_K, a_{K+1}) = (1.009, 1.013, 1.056, 1.199, 0.935)$ (for $-1dB$ trained model). As we expected, the trained weights in the later bits are larger. Also, the weight at the $K+1$th bit position is small because last bit is always zero and does not convey any information. On the other hand, trained weights in the beginning positions are small because these bits are naturally more robust to noise due to the sequential structure in Phase 2.

Result. The resulting BER curve is shown in Figure 6(-o-). We can see that the BER is noticeably decreased. In Figure 7(-o-), we can see that the BER in the last bits are reduced, and we can also see that the BER in the first bits are increased, as expected. Our use of unequal power allocation across information bits is in-line with other approaches from information/coding theory [33], [34]. We call this neural code Deepcode.

Complexity. Complexity and latency, as well as reliability, are important metrics in practice, as the encoder and the decoder need to run in real time on mobile devices. Deepcode has linear encoding and decoding complexity $O(K)$, where K denotes the information block length. S-K scheme and sequential forward error correcting codes, such as turbo codes and convolutional codes, also have linear encoding and decoding complexity. On the other hand, polar codes have encoding and decoding complexity $O(K \log K)$ [35]. General LDPC codes have encoding complexity $O(K^2)$ and decoding complexity $O(K)$, where as some optimized LDPC codes have encoding time complexity $O(K)$ [36].

Actual latencies are very hard to compare because the latency critically depends on how operations are implemented in the hardware. Turbo decoder, for example, is a belief-propagation decoder with multiple (e.g., 10 – 20) *iterations* of a component decoder, and each iteration is followed by a permutation. On the other hand, the

decoder for Deepcode is a 2-layered bi-directional GRU decoder, each with 50 hidden units, all of which are matrix multiplications that can be parallelized.

Whereas we can not compare the runtimes of Deepcode decoder and turbo decoder, we can compare the number of multiplications required per each bit. The bottleneck in the Deepcode decoder is the update of the write gate, forget gate, and the hidden state in 4 GRUs (forward GRU and backward GRU in layers 1 and 2); hence, in total, it includes 12 matrix (dimension 50×50) – vector (length 50) multiplications per bit. On the other hand, turbo decoder is a 10 – 20 iteration of BCJR algorithms, each of which includes between 10 to 100 multiplications per bit depending on the trellis used for the turbo code, followed by a permutation. We can similarly compare the encoder complexity. The bottleneck in the Deepcode encoder is an update of the hidden state in the RNN; which requires a matrix (dimension 50×50) – vector (length 50) multiplication per bit. Turbo encoder generates two recursive systematic convolutional codes, each requires 10s of boolean XORs per bit depending on the trellis, and a permutation of the message bit sequence. In the current form, Deepcode requires more computation than turbo code. Ideas such as knowledge distillation [37] and network binarization [38] can be used to potentially further reduce the complexity of the network. The optimization of Deepcode is beyond the scope of this paper and is left as a future work. We again note that the runtime comparison is open (e.g., matrix-vector multiplications can be highly parallelized).

IV. Practical considerations: noise and delay in feedback, finite precision, and blocklength

We considered so far the AWGN channel with noiseless output feedback with a unit time-step delay. In this section, we demonstrate the robustness of Deepcode (and its variants) under two variations on the feedback channel, *noise* and *delay*, as well as *finite precision*. We also present a generalization to longer block lengths. We show that (a) Deepcode and its variant that allows a K-step delayed feedback are more reliable than the state-of-the-art schemes in channels with *noisy* feedback, and (b) Deepcode concatenated with turbo code achieves superior error rate decay as block length increases with noisy feedback.

Noisy feedback. We show that Deepcode, trained on AWGN channels with *noisy* output feedback, achieves a significantly smaller BER than both S-K and C-L schemes [8]. In Figure 12 (Left), we plot the BER as a function of the feedback SNR for S-K scheme, C-L scheme, and Deepcode for a rate 1/3 code with 50 information bits, where we fix the forward channel SNR to be 0dB. As feedback SNR increases, we expect the BER to decrease. However, as shown in Figure 12 (Left), both C-L scheme, designed for channels with noisy feedback, and S-K scheme are sensitive to even a small amount of noise in the feedback, and reliability is almost independent of feedback quality. For C-L scheme, we take the experimental results for rate 1/3 (blocklength 5144) shown in [8].

Deepcode outperforms these two baseline (linear) codes by a large margin, with decaying error as feedback SNR increases, showing that Deepcode harnesses *noisy* feedback information to make communication more reliable. This is highly promising as the performance with noisy feedback is directly related to the practical communication channels. To achieve the performance shown in Figure 12, for example the line in red, training with matched SNR is required. For each datapoint, we use different neural codes specifically trained at the same SNR at the test noise. The neural encoder takes the feedback signal (as well as the message) as an input and includes power normalization tailored to the SNR of forward and feedback channels; hence, if trained with a mismatched SNR, the output of the neural code does not satisfy the power constraint. In Section V, we discuss how Deepcode differs depending on what SNR it was trained on, hence it is not universal.

Noise feedback with delay. We model the practical constraint of *delay* in the feedback, by introducing a variant of Deepcode that works with a K time-step delayed feedback (we refer to Appendix X for the details); recall K is the number of information bits and this code tolerates a large delay in the feedback. We see from Figure 12 (Left), that these neural codes are robust against delay in the feedback for noisy feedback channels of SNR up to 12dB.

Finite precision. We evaluate the sensitivity of Deepcode to the finite machine precision (without any re-training). In Figure 13, we plot the BER as a function of SNR for Deepcode implemented with a finite precision. We notice that under the 8 bit codeword quantization, Deepcode has almost no reliability loss. Re-training can potentially bring down the required precision even further. On the other hand, S-K scheme is very sensitive to the finite machine precision. In Figure 13, S-K scheme is implemented with 64-bit precision.

One of the reasons for the sensitivity is that its first transmission is a M-ary PAM where M represents the number of total messages (e.g., if information block length is 50, first transmission has to be done via a 2^{50}-PAM modulation). As a means to overcome this effect, one can consider using multiple blocks each of which

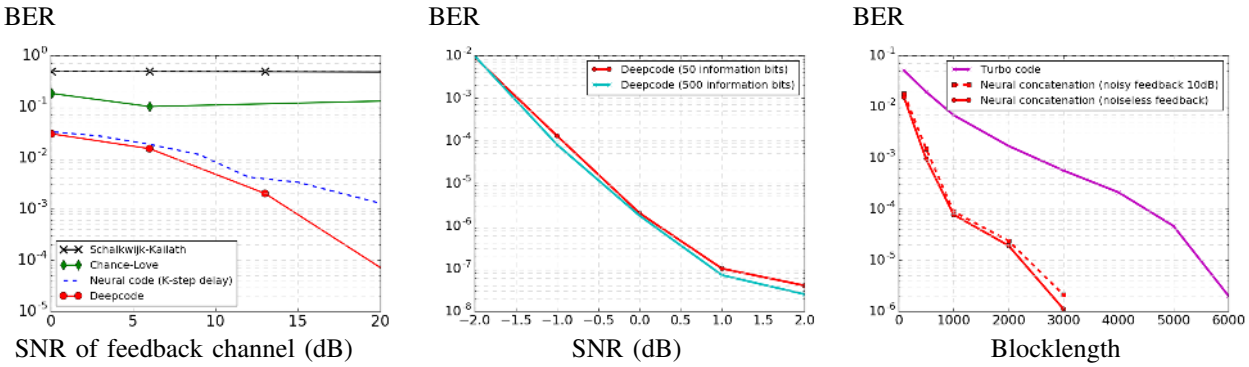


Fig. 12: (Left) Deepcode (introduced in Section III) and its variant code that allows K time-step delay significantly outperform the two baseline schemes in noisy feedback scenarios. (Middle) By unrolling the RNN cells of Deepcode, the BER of Deepcode remains unchanged for block lengths 50 to 500. (Right) Concatenation of Deepcode and turbo code (with and without noise in the feedback) achieves BER that decays exponentially as block length increases, faster than turbo codes (without feedback) at the same rate.

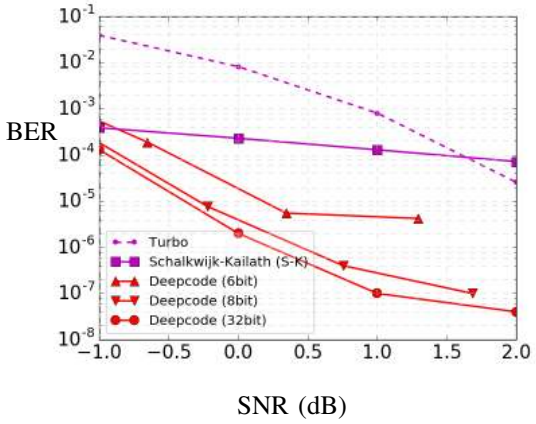


Fig. 13: Performance of Deepcode under the scenarios where codewords are quantized to 8 bits and 6 bits.

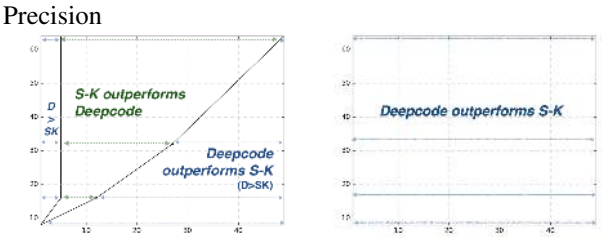


Fig. 14: Relative performance of S-K and Deepcode as a function of machine precision (y-axis) and the length of each coding block (x-axis) for noiseless feedback (Left) and noisy feedback (Right). When precision is small (e.g., 8 bit) or feedback is noisy (even a small amount of noise; e.g., feedback SNR is 40dB), Deepcode outperforms the S-K scheme. When precision is large enough and feedback channel is noiseless, S-K can outperform Deepcode by reducing the coding block length.

has a smaller block length (e.g., 5 blocks of length-10 codewords all together represent a length-50 codeword.) In Figure 14, we show the effect of precision (y-axis) and the length of each coding block (x-axis) on the relative performance of S-K and Deepcode for noiseless feedback (Left) and noisy feedback (Right). We let the forward SNR be 0dB for both cases and feedback SNR be 40dB (hence, very small noise) for the noisy feedback scenario. Figure 14 (Left) demonstrates that the S-K scheme can outperform Deepcode for noiseless feedback by reducing the coding block length as long as precision value is large enough; however, for a smaller precision (e.g., 8 bit), Deepcode always outperforms the S-K scheme. Figure 14 (Right) demonstrates that regardless of the precision and the length of each coding block, Deepcode always outperforms the S-K scheme.

Generalization to longer block lengths. In wireless communications, a wide range of blocklengths are of interest (e.g., 40 to 6144 information bits in LTE standards). In previous sections, we considered block length of 50 information bits. Here we show how to generalize Deepcode to longer block lengths and achieve an improved reliability as we increase the block length.

A natural generalization of the RNN-based Deepcode is to unroll the RNN cells. In Figure 12 (Middle), we plot the BER as a function of the SNR, for 50 information bits and length 500 information bits (with noiseless feedback) when we unroll the RNN cells. We can see that the BER remains the same as we increase block lengths. This is not an entirely satisfying generalization because, typically, it is possible to design a code for which error rate decays faster as block length

increases. For example, turbo codes have error rate decaying exponentially (log BER decays linearly) in the block length as shown in Figure 12 (Right). This critically relies on the interleaver, which creates long range dependencies between information bits that are far apart in the block. Given that Deepcode is a sequential code, there is no strong long range dependence. Each transmitted bit depends on only a few past information bits and their feedback (we refer to Section V for a detailed discussion).

To resolve this problem, we propose a new concatenated code which concatenates Deepcode (as inner code) and turbo code as an outer code. The outer code is not restricted to a turbo code, and we refer to Appendix XI for a detailed discussion. In Figure 12 (Right), we plot the BERs of the concatenated code, in channels with both noiseless and noisy feedback (of feedback SNR 10dB), and turbo code, both at rate $1/9$ at (forward) SNR -6.5dB. From the figure, we see that even with noisy feedback, BER drops almost exponentially (log BER drops linearly) as block length increases, and the slope is sharper than the one for turbo codes. We also note that in this setting, C-L scheme suggests not using the feedback.

V. Interpretation

Thus far we have used information theoretic insights in driving our deep learning designs. Here, we ask if the deep learning architectures we have learnt can provide an insight to the information theory of communications with feedback. We aim to understand the behavior of Deepcode (i.e., how coded bits are generated via RNN in Phase 2). We show that in the second phase, (a) the encoder focuses on refining information bits that were corrupted by large noise in the first phase; and (b) the coded bit depends on past as well as current information bits, i.e., coupling in the coding process.

Correcting noise from previous phase. The main motivation behind the proposed two-phase encoding scheme is to use the Phase 2 to clean the noise added in Phase 1. The encoder at Phase 2 knows how much noise was added in Phase 1 (exactly if noiseless feedback and approximately if noisy). Potentially, it could learn to send this information in the Phase 2, so that the decoder can refine the corrupted information bits sent in Phase 1. Interpreting the parity bits confirms this conjecture as shown in Figure 15. We show as a scatter plot multiple instances of the pairs of random variables $(n_k, c_{k,1})$ (left) and $(n_k, c_{k,2})$ (right), where n_k denotes the noise added to the transmission of b_k in the first phase. We are plotting 1,000 sample points: 20 samples for each k and for $k \in \{1, \ldots, 50\}$. This illustrates how the encoder has learned to send rectified linear unit (ReLU(x)=$\max\{0, x\}$) functional of the noise n_k to send the noise information while efficiently using the power. Precisely, the dominant term in the parity bit can be closely approximated by $c_{k,1} \simeq -(2b_k - 1) \times \mathrm{ReLU}(-n_k(2b_k - 1))$, and $c_{k,2} \simeq (2b_k - 1) \times \mathrm{ReLU}(-n_k(2b_k - 1))$.

Consider the case when $b_k = 1$. If the noise added to bit b_k in Phase 1 is positive, then the bit is likely to have been correctly decoded, and the parity chooses not to send any information about n_k. The encoder generates coded bits close to zero (i.e., does not further refine b_k). Otherwise, the encoder generates coded bits proportional to the noise n_k, i.e., uses more power to refine b_k.

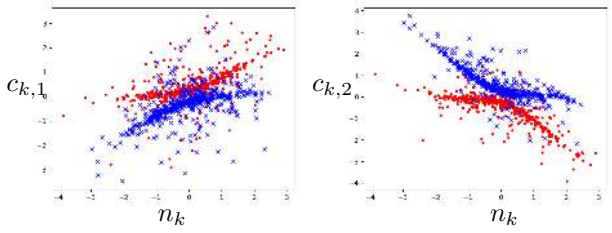


Fig. 15: Noise in first phase n_k vs. first parity bit $c_{k,1}$ (left) and second parity bit $c_{k,2}$ (right) under noiseless feedback channel and forward AWGN channel of SNR 0dB. Blue 'x' data points correspond to those samples conditioned on $b_k = 1$ and red 'o' points correspond to those samples conditioned on $b_k = 0$.

Ideally, for practical use, we want to use the same code for a broad range of varying SNR, as we might be uncertain about the condition of the channel we are operating on. This is particularly true for code for non-feedback channels. For channels with output feedback, however, all known encoding schemes adapt to the channel. For instance, in S-K scheme, in order to achieve the optimal error rate, it is critical to choose the optimal power allocation across transmission symbols depending on the (forward) channel SNR. Similarly, C-L scheme also requires pre-computation of optimal power allocation across transmission depending on the forward and feedback SNRs. In the case of AWGN channels with feedback, it is not even clear how one could meet the power constraints, if not adapting to the channel SNR.

In Figure 16, we show how the trained Deepcode has learned to adapt to the channel conditions. For various choices of forward channel $\mathrm{SNR_f}$ and feedback channel $\mathrm{SNR_{fb}}$, each scatter plot is showing 5,000 sample points: 100 samples for each k and for $k \in \{1, \ldots, 50\}$. On the top row, as forward signal power decreases, the parity gradually changes from $c_{k,1} \simeq -(2b_k - 1) \times \mathrm{ReLU}(-n_k(2b_k - 1))$ to $c_{k,1} \simeq (2b_k - 1) - n_k$. On the bottom row, as feedback noise increases (i.e., feedback SNR decreases), the parity gradually becomes less correlated with the sum of forward and feedback

noises $n_k + w_k$. Note that under noisy feedback, n_k is not available to the encoder, but $n_k + w_k$ is what is available to the encoder ($n_k + w_k = \tilde{y}_k - c_k$).

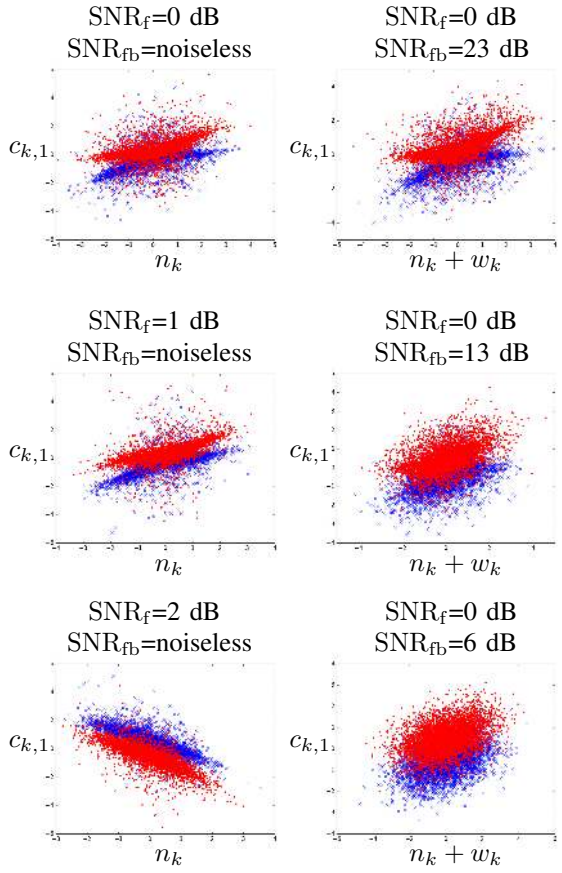


Fig. 16: Noise in first phase n_k vs. first parity bit $c_{k,1}$ as forward SNR increases for noiseless feedback (left column) and sum of forward and feedback noises in first phase $n_k + w_k$ vs. first parity bit $c_{k,1}$ as feedback SNR decreases for fixed forward SNR 0dB (right column). Blue 'x' data points correspond to those samples conditioned on $b_k = 1$ and red 'o' points correspond to those samples conditioned on $b_k = 0$.

Coupling. A natural question is whether our feedback code is exploiting the memory of RNN and coding information bits jointly. To answer this question, we look at the correlation between information bits and the coded bits. If the memory of RNN were not used, we would expect the coded bits $(c_{k,1}, c_{k,2})$ depend only on b_k. We find that $\mathbb{E}[c_{k,1}b_k] = -0.42, \mathbb{E}[c_{k,1}b_{k-1}] = -0.24, \mathbb{E}[c_{k,1}b_{k-2}] = -0.1, \mathbb{E}[c_{k,1}b_{k-3}] = -0.05$, and $\mathbb{E}[c_{k,2}b_k] = 0.57, \mathbb{E}[c_{k,2}b_{k-1}] = -0.11, \mathbb{E}[c_{k,2}b_{k-2}] = -0.05, \mathbb{E}[c_{k,2}b_{k-3}] = -0.02$ (for the encoder for forward SNR 0dB and noiseless feedback). This result implies that the RNN encoder does make use of the memory, of length two to three.

Overall, our analysis suggests that Deepcode exploits memory and selectively enhances bits that were subject to larger noise - properties reminiscent of any good code. We also observe that the relationship between the transmitted bit and previous feedback demonstrates a *non-linear* relationship as expected. Thus our code has features requisite of a strong feedback code. Furthermore, improvements can be obtained if instead of transmitting two coded symbols per bit during Phase 2, an attention-type mechanism can be used to zoom in on bits that were prone to high noise in Phase 1. These insights suggest the following generic feedback code: it is a sequential code with long cumulative memory but the importance of a given bit in the memory is *dynamically* weighted based on the feedback.

VI. System and Implementation Issues

We began with the idealized Shannon model of feedback and have progressively considered practical variants (delay, noise and active feedback). In this section we extend this progression by studying design decisions in real-world implementations of Deepcode (our neural-network feedback-enabled codes). We do this in the context of cellular wireless systems, with specific relevance to the upcoming 5G LTE standard.

LTE cellular standards prescribe separate uplink and downlink transmissions (usually in frequency division duplex mode). Further, these transmissions are scheduled in a centralized manner by the base station associated with the cell. In many scenarios, the traffic flowing across uplink and downlink could be asymmetric (example: more "downloads" than "uploads" leads to higher downlink traffic than the combined uplink ones). In such cases, there could be more channel resources in the uplink than the traffic demand. Given the sharp inflexible division among uplink and downlink, these resources go unused. We propose to link, *opportunistically*, unused resources in one direction to aid the reliability of transmission in the opposite direction – this is done via using the feedback codes developed in this paper. Note that the availability of such unused channel resources is known in advance to the base station which makes scheduling decisions on *both* directions of uplink and downlink – thus such a synchronized cross uplink-downlink scheduling is readily possible.

The availability of the feedback traffic channel enables the usage of the codes designed in this paper – leading to much stronger reliability than the feedforward codes alone. Combined with automatic repeat request (ARQ), this leads to fewer retransmissions and smaller average transmission time than the traditional scheme of feedforward codes combined with ARQ would achieve. In order to numerically evaluate the expected benefits of such a system design, in Figure 17, we plot BLER as

a function of number of (re)transmissions for Deepcode under noiseless and noisy feedback and feedforward codes (for a rate $1/3$ code with 50 information bits). From this figure, we can see that combining Deepcode with ARQ allows fewer block transmissions to achieve the target BLER compared to the state-of-the-art codes. The performance of Deepcode depends on the quality of the feedback channel. As feedback channel becomes less noisy, Deepcode requires fewer retransmissions. We note that in measuring the BLER of neural code under noisy feedback (10dB), we used a variant of Deepcode, shown as Act-Deepcode, which allows an active feedback of rate $3/4$; in Phase 1, the decoder sends back RNN encoded bits at rate 1/2. Phase 2 works as in Deepcode. Hence, for a rate 1/3 code with 50 information bits, the decoder makes 200 usages of the feedback channel (204 with zero padding). Improving further the performance of (active) Deepcode at realistic feedback SNRs (such as 10dB or lower) is an important open problem. The improvements could come from architectural or learning methodology innovations or a combination of both.

We propose using Deepcode when the feedback SNR is high. Practically, a user may not always have a high SNR feedback channel, but when there are multiple users, it is possible that some of the users have high SNR feedback channels. For example, in scenarios where a base station communicates with multiple users, we propose scheduling users based on their feedback as well as forward channel qualities, utilizing multiuser diversity. In Internet-of-Things (IoT) applications, feedback channel SNR can be much higher than forward SNR; e.g., a small device with limited power communicates a message to the router connected to the power source.

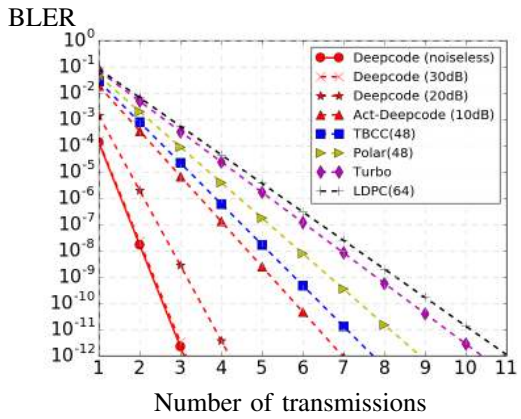


Fig. 17: BLER as a function of number of transmissions for a rate 1/3 code with 50 information bits where forward SNR is 0dB. Deepcode allows fewer transmissions than feedforward codes to achieve the target BLER.

VII. Conclusion

In this paper we have shown that appropriately designed and trained RNN codes (encoder and decoder), which we call Deepcode, outperform the state-of-the-art codes by a significant margin on the challenging problem of communicating over AWGN channels with noisy output feedback, both on the theoretical model and with practical considerations taken into account. By concatenating Deepcode with a traditional outer code, the BER curve drops significantly with increasing block lengths, allowing generalizations of the learned neural network architectures. The encoding and decoding capabilities of the RNN architectures suggest that new codes could be found in other open problems in information theory (e.g., network settings), where practical codes are sorely missing. Our work suggests several immediate avenues continue the research program begun by this paper, solutions to which will have significant practical impacts.

Learning to take advantage of the block lengths. The first one is an interesting new challenge for machine learning, that has not been posed before to the best of our knowledge. We proposed concatenation in Section IV to achieve the block-length gain. By concatenating Deepcode with a traditional inner code, the BER curve drops significantly with increasing block lengths, allowing generalizations of the learned neural network architectures. However, concatenation comes at the cost of reduced rate.

A more natural way to achieve the block-length gain is to incorporate structures of the modern codes, in particular turbo codes. Turbo codes use interleavers to introduce long range dependency on top of standard convolutional codes, achieving error rate that exponentially decays in block-lengths as desired. In the encoder, we can easily include an inter-leaver with two neural encoders we proposed. However, the major challenge is in decoding. Turbo decoder critically relies on BCJR decoder's accurate estimate of the posterior probability of each information bit. This is in turn fed into the next phase of turbo decoder, which refines the likelihood iteratively. For the proposed neural feedback code, there exists no decoder that can output accurate posterior likelihood. Further, there exists no decoder that can take as part of the input the side information from the previous phase on the prior likelihood of the information bit. This poses an interesting challenge for deep learning. As a means to overcome this challenge, [39] proposes a novel architecture which harmoniously combines the interleaving idea of turbo code and iterative decoding together with CNN architectures. It is demonstrated that the neural code in [39] exhibits the blocklength gain (i.e., the error rate decays as block length increases).

Interpreting Deepcode. The second challenge is in using the lessons learned from the trained Deepcode to contribute back to the communication theory. We identified in Section V some parts of the parity symbols of the trained Deepcode. However, how Deepcode is able to exploit the feedback symbols remains mysterious, despite our efforts to interpret the trained neural network. It is an interesting challenge to disentangle the neural encoder, and provide a guideline for designing simple feedback encoders that enjoy some of the benefits of the complex neural encoder. Manually designing such simple encoders without training can provide a new family of feedback encoders that are simple enough to be mathematically analyzed.

Rate beyond 1/3. The third challenge is to generalize Deepcode to rates beyond 1/3. Our neural code structure can be immediately generalized to rates $1/r$ for $r = 2, 3, 4, \cdots$. For example, we have preliminary results showing that a rate-$1/2$ RNN based feedback code beats the state-of-the-art codes for short block lengths (e.g., 64) under low SNRs (e.g., below 2dB). Extensive experiments and simulations over various rates and comparison to state-of-the-art codes are yet to be explored. On the other hand, generalization to rates higher than $1/2$ requires a new architecture of encoders. In this direction, we propose two potential approaches. One is to use a higher-order modulation (e.g., pulse amplitude modulation) and generate parity bits for super symbols which are functions of multiple information bits. The other is to use puncturing, a widely used technique to design high rate codes from low rate codes (e.g., convolutional codes); the encoder first generates a low rate code and then throws away some of the coded bits and sends only a fraction of the coded bits. Generalization to higher rate codes via these two approaches is of great practical interest.

ACKNOWLEDGEMENT

We thank Shrinivas Kudekar and Saurabh Tavildar for helpful discussions and providing references to the state-of-the-art feedforward codes. We thank Dina Katabi for a detailed discussion that prompted the work on system implementation.

REFERENCES

[1] C. E. Shannon, "A mathematical theory of communication, part i, part ii," *Bell Syst. Tech. J.*, vol. 27, pp. 623–656, 1948.

[2] C. J. Il, J. Mayank, S. Kannan, L. Philip, and K. Sachin, "Achieving single channel, full duplex wireless communication," in *Proceedings of the 16th Annual International Conference on Mobile Computing and Networking, MOBICOM 2010, Chicago, Illinois, USA, September 20-24, 2010*.

[3] C. Shannon, "The zero error capacity of a noisy channel," *IRE Transactions on Information Theory*, vol. 2, no. 3, pp. 8–19, 1956.

[4] J. Schalkwijk and T. Kailath, "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, 1966.

[5] J. Schalkwijk, "A coding scheme for additive noise channels with feedback–ii: Band-limited signals," *IEEE Transactions on Information Theory*, vol. 12, no. 2, pp. 183–189, April 1966.

[6] R. G. Gallager and B. Nakiboglu, "Variations on a theme by Schalkwijk and Kailath," *IEEE Transactions on Information Theory*, vol. 56, no. 1, pp. 6–17, Jan 2010.

[7] N. C. Martins and T. Weissman, "Coding for additive white noise channels with feedback corrupted by quantization or bounded noise," *IEEE Transactions on Information Theory*, vol. 54, no. 9, pp. 4274–4282, Sep. 2008.

[8] Z. Chance and D. J. Love, "Concatenated coding for the AWGN channel with noisy feedback," *IEEE Transactions on Information Theory*, vol. 57, no. 10, pp. 6633–6649, Oct 2011.

[9] Y.-H. Kim, A. Lapidoth, and T. Weissman, "The Gaussian channel with noisy feedback," in *IEEE International Symposium on Information Theory*. IEEE, 2007, pp. 1416–1420.

[10] P. Elias, "Coding for noisy channels," in *IRE Convention record*, vol. 4, 1955, pp. 37–46.

[11] S. C. Draper and A. Sahai, "Noisy feedback improves communication reliability," in *2006 IEEE International Symposium on Information Theory*, July 2006, pp. 69–73.

[12] S. C. Draper and A. Sahai, "Variable-length channel coding with noisy feedback," *European Transactions on Telecommunications*, vol. 19, pp. 355–370, 2008.

[13] A. Ben-Yishai and O. Shayevitz, "Interactive schemes for the AWGN channel with noisy feedback," *IEEE Transactions on Information Theory*, vol. 63, no. 4, pp. 2409–2427, April 2017.

[14] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, Dec 2017.

[15] A. Felix, S. Cammerer, S. Dörner, J. Hoydis, and S. t. Brink, "OFDM-autoencoder for end-to-end learning of communications systems," in *IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, June 2018, pp. 1–5.

[16] S. Cammerer, S. Dörner, J. Hoydis, and S. ten Brink, "End-to-end learning for physical layer communications," in *The International Zurich Seminar on Information and Communication (IZS 2018) Proceedings*. ETH Zurich, 2018, pp. 51–52.

[17] T. J. O'Shea, T. Erpek, and T. C. Clancy, "Deep learning based MIMO communications," *CoRR*, vol. abs/1707.07980, 2017.

[18] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention," in *IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, 2016, pp. 223–228.

[19] N. Farsad, M. Rao, and A. Goldsmith, "Deep learning for joint source-channel coding of text," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[20] H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, and P. Viswanath, "Communication algorithms via deep learning," in *The International Conference on Representation Learning (ICLR 2018) Proceedings*. Vancouver, April, 2018.

[21] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *IEEE 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton) 2016*, pp. 341–346.

[22] 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 Journal of Selected Topics in Signal Processing*, 2018.

[23] X. Tan, W. Xu, K. Sun, Y. Xu, Y. Be'ery, X. You, and C. Zhang, "Improving massive MIMO message passing detectors with deep neural network," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 1267–1280, Feb 2020.

[24] W. Lyu, Z. Zhang, C. Jiao, K. Qin, and H. Zhang, "Performance evaluation of channel decoding with deep neural networks," *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2018.

[25] M. Lian, C. Häger, and H. D. Pfister, "What can machine learning teach us about communications?" in *IEEE Information Theory Workshop (ITW)*, Nov 2018, pp. 1–5.

[26] F. Carpi, C. Häger, M. Martalò, R. Raheli, and H. D. Pfister, "Reinforcement learning for channel coding: Learned bit-flipping decoding," in *57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Sep. 2019, pp. 922–929.
[27] J. Kosaian, K. Rashmi, and S. Venkataraman, "Learning a code: Machine learning for approximate non-linear coded computation," *arXiv preprint arXiv:1806.01259*, 2018.
[28] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.
[29] Y. Polyanskiy, H. V. Poor, and S. Verdú, "Channel coding rate in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2307–2359, 2010.
[30] H. Huawei, "Performance evaluation of channel codes for control channel," *3GPP TSG-RAN WG1 #87 Reno, U.S.A., November 14-18, 2016*, vol. R1-1611257. [Online]. Available: www.3gpp.org/ftp/tsg_ran/WG1_RL1/TSGR1_87/Docs/R1-1611257.zip
[31] T. Erseghe, "On the evaluation of the Polyanskiy-Poor-Verdu converse bound for finite block-length coding in AWGN," *IEEE Transactions on Information Theory*, vol. 61, January 2014.
[32] Y. H. Kim, A. Lapidoth, and T. Weissman, "The Gaussian channel with noisy feedback," in *IEEE International Symposium on Information Theory*, June 2007, pp. 1416–1420.
[33] T. Duman and M. Salehi, "On optimal power allocation for turbo codes," in *IEEE International Symposium on Information Theory - Proceedings*. IEEE, 1997, p. 104.
[34] H. Qi, D. Malone, and V. Subramanian, "Does every bit need the same power? an investigation on unequal power allocation for irregular LDPC codes," in *2009 International Conference on Wireless Communications Signal Processing*, Nov 2009, pp. 1–5.
[35] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015.
[36] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, pp. 638 – 656, 03 2001.
[37] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015.
[38] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.
[39] H. A. S. O. S. K. P. V. Yihan Jiang, Hyeji Kim, "Feedback turbo autoencoder," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020.
[40] G. D. Forney, Jr, "Concatenated codes," *MIT Press, Cambridge, MA.*, 1966.
[41] K. Miwa, N. Miki, T. Kawamura, and M. Sawahashi, "Performance of decision-directed channel estimation using low-rate turbo codes for dft-precoded OFDMA," in *IEEE 75th Vehicular Technology Conference (VTC Spring)*, May 2012, pp. 1–5.

Hyeji Kim Hyeji Kim is a researcher at Samsung AI Research Cambridge. Prior to joining Samsung AI Research in 2018, she was a postdoctoral research associate at the university of Illinois at Urbana Champaign. She received her Ph.D. and M.S. degrees in Electrical Engineering from Stanford University in 2016 and 2013, respectively, and her B.S. degree with honors in Electrical Engineering from KAIST in 2011. She is a recipient of the Stanford Graduate Fellowship and participant of the Rising Stars in EECS Workshop in 2015.

Yihan Jiang Yihan Jiang is a Ph.D. candidate at the Electrical and Computer Engineering Department, University of Washington, Seattle at Washington. He received his M.S degree in 2014 in electrical and computer engineering from UC San Diego and a B.S degree from Beijing Institute of Technology in 2012. His research interests are in the areas of channel coding, information theory, deep learning, and federated learning.

Sreeram Kannan Sreeram Kannan is currently an assistant professor at University of Washington, Seattle. He was a postdoctoral scholar at University of California, Berkeley between 2012-2014 before which he received his Ph.D. in Electrical Engineering and M.S. in mathematics from the University of Illinois Urbana Champaign. He is a recipient of the 2019 UW ECE outstanding teaching award, 2017 NSF Faculty Early CAREER award, the 2013 Van Valkenburg outstanding dissertation award from UIUC, a co-recipient of the 2010 Qualcomm Cognitive Radio Contest first prize, a recipient of 2010 Qualcomm (CTO) Roberto Padovani outstanding intern award, a recipient of the SVC Aiya medal from the Indian Institute of Science, 2008, and a co-recipient of Intel India Student Research Contest first prize, 2006. His research interests include the applications of information theory and learning to blockchains, computational biology and wireless networks.

Sewoong Oh Sewoong Oh is an Associate Professor in the Paul G. Allen School of Computer Science & Engineering at the University of Washington. Previous to joining University of Washington in 2019, he was an Assistant Professor in the department of Industrial and Enterprise Systems Engineering at University of Illinois at Urbana-Champaign since 2012. He received his PhD from the department of Electrical Engineering at Stanford University in 2011, under the supervision of Andrea Montanari. Following his PhD, he worked as a postdoctoral researcher at Laboratory for Information and Decision Systems (LIDS) at MIT, under the supervision of Devavrat Shah. Sewoong's research interest is in theoretical machine learning. He was co-awarded the ACM SIGMETRICS best paper award in 2015, NSF CAREER award in 2016, ACM SIGMETRICS rising star award in 2017, and GOOGLE Faculty Research Award in 2017 and 2020.

Pramod Viswanath Pramod Viswanath is a professor of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign. He received his Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley in 2000. His current research interests include blockchain technologies from a variety of angles: networking protocols, consensus algorithms, payment channels, distributed coded storage and incentive designs. He is a co-founder and CEO of Applied Protocol Research, a startup focused on developing core blockchain technologies. He has received the Eliahu Jury Award, the Bernard Friedman Prize, a NSF CAREER award, and the Best Paper Award at the Sigmetrics conference in 2015. He is a co-author, with David Tse, of the text Fundamentals of Wireless Communication, which has been used in over 60 institutions worldwide.

APPENDIX

VIII. STATE-OF-THE ART CODES USED IN COMPARISON

In this section, we provide details on how to compute the BER and BLER of state-of-the art feedforward codes. LTE turbo code used in the simulation uses trellis-([13, 15], 13) convolutional code (octal notation) as a component code, and uses quadratic permutation polynomial (QPP) interleaver. Decoding is done by 8 iterations of Belief Propagation (BP) decoder that uses a posteriori probability (APP) decoder as the constituent decoder. Tail-bitting convolutional codes (TBCC) used in the simulation has a constraint length 7 and trellis ([123,135,157]) (in octal notation), and uses Viterbi decoder. Polar code used in the simulation uses successive cancellation list decoding (SCL) with list size 8. LDPC code used in the simulation (Rate 1/3, maps 64 bits to a length-196 codeword with sub-matrix dimension 16) uses the parity check matrix shown below, and layered offset min-sum decoder is used with offset parameter 0.22 and (max) iteration 25.

$$\begin{pmatrix} 10 & 11 & 2 & 3 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 15 & 9 & 9 & 14 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\ 6 & -1 & 5 & 13 & -1 & 11 & 0 & -1 & -1 & -1 & -1 & -1 \\ -1 & 5 & -1 & 8 & 12 & -1 & 6 & 0 & -1 & -1 & -1 & -1 \\ -1 & 11 & -1 & -1 & 1 & -1 & -1 & 11 & 0 & -1 & -1 & -1 \\ -1 & 2 & -1 & -1 & 14 & 12 & -1 & 7 & -1 & 0 & -1 & -1 \\ -1 & 15 & 10 & -1 & -1 & -1 & -1 & -1 & 11 & -1 & 0 & -1 \\ -1 & -1 & -1 & 7 & -1 & 11 & -1 & 3 & -1 & -1 & -1 & 0 \end{pmatrix}$$

IX. IMPLEMENTATION DETAILS

In this section, we provide implementation details on the neural encoders and decoders introduced in Section III together with an illustration of the end-to-end communication system in Figure 18.

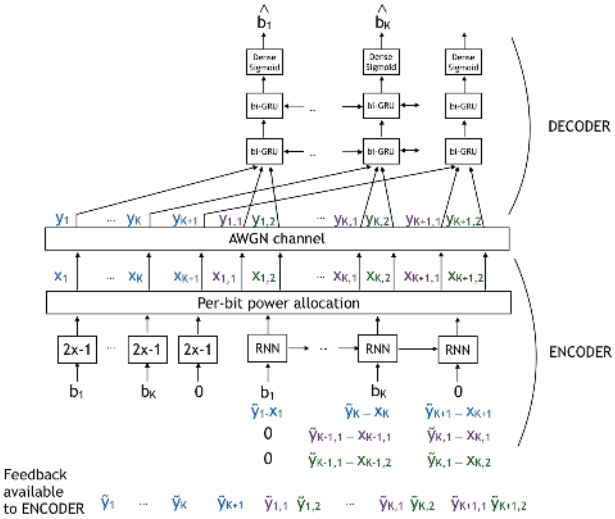


Fig. 18: Illustration of communications system with Deepcode

X. ILLUSTRATION ON SCHEME FOR DELAYED FEEDBACK.

Practical feedback typically is delayed for a random time, thus the encoder cannot use immediate feedback to encode. The feedback is randomly delayed up to block length K, we are restricted not to use feedback till K bits are transmitted.

We propose a delayed feedback scheme to overcome noisy feedback and delaying effect; the 1/3 code rate encoder is shown in Figure 19. In the first phase, the K information bits can be encoded by Bi-GRU, while the feedback is delayed and can only be used in the next phase. The second and third phases use uni-directional GRU to encode with K-delayed feedback. For example, at index m of phase 2, the encoder can only use the feedback before index m of phase 1. The decoder is a Bi-GRU which waits to decode until all transmissions are received, same as in Deepcode.

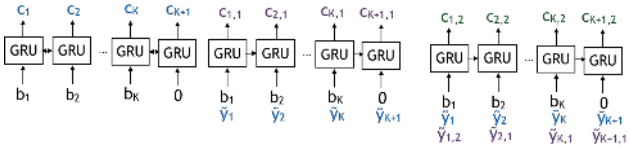


Fig. 19: Encoder for delayed feedback

XI. CONCATENATION OF DEEPCODE WITH EXISTING CODES

Concatenated codes are constructed from two or more codes, originally proposed by Forney [40]. We concatenate forward error correcting codes (that does not use a feedback) with our Deepcode that makes use of feedback. Encoding is performed in two steps; we first map information bits into a turbo code, and then encode the turbo code via an encoder for channels with feedback. Decoding is also performed in two steps. In the first step, decoder recovers the estimates of turbo codes. In the second step, the decoder recovers information bits based on the estimates of turbo codes. For the experiment in Section IV, for which results are shown in Figure 12 (Right), we use the rate 1/3 LTE turbo code as an outer code; LTE turbo code uses ([13, 15], 13) convolutional code (octal notation) as a component code. We compare the performance of the concatenated code with a rate 1/9 turbo code, which uses ([13,17,16,15,11],13) convolutional code as a component code (introduced in [41]). Besides turbo codes, any existing codes (e.g., LDPC, polar, convolutional codes) can be used as an outer code. We also note that C-L scheme is based on the concatenation idea [8].