Physical Layer Communication via Deep Learning

Hyeji Kim, Sewoong Oh, and Pramod Viswanath

***Abstract*—Reliable digital communication is a primary workhorse of the modern information age. The disciplines of communication, coding, and information theories drive the innovation by designing efficient codes that allow transmissions to be robustly and efficiently decoded. Progress in near optimal codes is made by individual human ingenuity over the decades, and breakthroughs have been, befittingly, sporadic and spread over several decades.**

Deep learning is a part of daily life where its successes can be attributed to a lack of a (mathematical) generative model. Deep learning empirically fits neural network models to the data, and the result has been extremely potent. In yet other applications, the data is generated by a simple model and performance criterion mathematically precise and training/test samples infinitely abundant, but the space of algorithmic choices is enormous (example: chess). Deep learning has recently shown strong promise in these problems too (example: alphazero). The latter scenario is a good description of communication theory. The mathematical models underlying canonical communication channels allow one to sample an unlimited amount of data to train and test the communication algorithms ((encoder, decoder) pairs) and the metric of bit (or block) error rate allows for mathematically precise evaluation.

Motivated by the successes of deep learning in mathematically well defined and extremely challenging tasks of chess, Go, and protein folding, we posit that deep learning methods can play a crucial role in solving core goals of coding theory: designing new (encoder, decoder) pairs that improve state of the art performance over canonical channel models. This manuscript surveys recent advances towards demonstrating this hypothesis, by focusing on strengthening a specific family of coding methods – sequential codes (convolutional codes) and codes that use them as basic building blocks (Turbo codes) – via deep learning methods. New state of the art results are derived on several canonical channels, including the AWGN channel with feedback.

I. Introduction

Reliable communication, both wireline (ethernet, cable and DSL modems) and wireless (cellular, Wifi, and satellite), is a primary workhorse of the modern information age. At the core of the physical layer communication system is the design of codes that a can be robustly transmitted and efficiently decoded under noisy conditions. In this discipline of communication theory over the past 70 years, breakthroughs are marked by the discoveries of novel codes including convolutional codes, Turbo codes, low density parity check (LDPC) codes and polar codes. The impact on humanity is enormous – every cellular phone designed uses one of these codes, which feature in global cellular standards ranging from the 2nd generation to the 5th generation respectively, and are text book material.

The canonical setting is a point-to-point channel with the additive white Gaussian noise (AWGN), and performance of a code in this setting is the gold standard. This naturally fits much of wireline and wireless communications although the front end of the receiver may have to be specifically designed before being processed by the decoder (example: intersymbol equalization in cable modems, beamforming and sphere decoding in multiple antenna wireless systems). In communication theory, the two long term goals are: (a) design of new, computationally efficient, codes that improve the state of the art (probability of correct reception) over the AWGN setting. Although the current codes already operate close to the (infinite block length) information theoretic "Shannon limit", there is room to improve the performance in some regimes (at "moderate" block lengths, high rates). There is also an emphasis on robustness and adaptability to deviations from the AWGN settings (such as in urban, pedestrian, vehicular settings). (b) design of new codes for multi-terminal (i.e., beyond point-to-point) settings – examples include the feedback channel, the relay channel and the interference channel.

Progress over these long term goals has generally been driven by individual human ingenuity and, befittingly, is sporadic. For instance, the time duration between convolutional codes (2nd generation cellular standards) to polar codes (5th generation cellular standards) is over 4 decades. Deep learning is fast emerging as capable of learning sophisticated algorithms from observed data (input, action, output) alone and has been remarkably successful in a large variety of human endeavors. Deep learning is already a part of daily life, owing to its successes in computer vision [1] and natural language processing [2]. In these applications, the success of the model-free deep learning approach can be attributed to a lack of (mathematical) generative model. In yet other applications, the data is generated by a simple model and performance criterion mathematically precise and training/test samples infinitely abundant, but the space of algorithmic choices is enormous (example: chess). Deep learning has recently shown strong promise in these problems too (example: AlphaZero [3]). Motivated by these successes, we envision that deep learning methods can play a crucial role in improving the state of the art in both the aforementioned goals of communication theory.

Communication and coding theory deals with problems of the latter type. The core questions are studied in the context of simple mathematical models (example: AWGN channel mentioned earlier). The performance metric is also mathematically precise (example: bit error rate (BER) vs signal to noise ratio (SNR) curve). The key complexity is algorithmic: the space of efficient encoders and decoders is simply too large and a priori unstructured. The recent success of deep learning on mathematically well defined problems (games such as Chess and Go, protein folding) leads us to hypothesize similar breakthroughs in communication theory. This tutorial summarizes the recent progress (and strong promise) along this hypothesis.

Traditional communication methods involve imposing some structure to encoders (example: linearity of codes) and decoders (example: iterative decoding); this pares down the com-

plexity of space of encoders and decoders. These structures have entailed tremendous human effort and are the product of substantial ingenuity; Figure 1 illustrates a taxonomy within the linearity structures of codes; the structures allow for computationally efficient decoding. This paper focuses on one specific family of codes – sequential codes – and aims to enlargen and strengthen this family of codes. This enlargening of the (encoder, decoder) pair is possible by a family of neural networks known as *recurrent neural networks* (RNNs). We will see a variety of canonical contexts in which the enlargened family strictly improves on the state of the art (among all known codes).

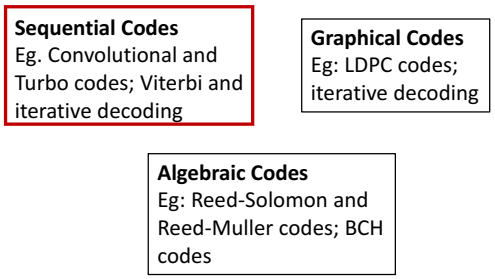


Fig. 1: Taxonomy of different coding methods. We focus on strengthening the class of sequential codes (convolutional codes and Turbo codes that are built using them) via RNNs.

We begin this exploration with a natural question: we fix the encoder to be a specific sequential code (convolutional code) and ask whether the classical ML decoder (Viterbi decoding) can be "learnt from data alone"; this is the focus of § IV. We see that creatively trained RNNs can successfully recover the Viterbi algorithm and the Turbo decoder; the key intellectual contribution is in the identification of using the "hardest" SNR (signal to noise ratio) to train the neural network, despite the decoder being tested on a variety of SNRs and bit (or block) error rates being empirically evaluated.

Next we describe recent successes in discovering new communication algorithms that strictly improve upon the state of the art. In § V-A, we study the design of new decoders based on RNNs that successfully outperform state of the art decoders for Turbo codes on non-AWGN channels. An overarching theme in each of the works we summarize is the utilization of recent advances in deep learning, but being mindful that training such models poses several challenges distinct to communication systems. The exponential size of the codebook, generalization in block lengths, availability of infinite training samples all pose novel challenges and opportunities. Successful training and testing crucially relies on innovations on (a) what architecture to pick; (b) what training samples/SNR to use; and (c) what optimization procedures to follow, including loss functions and regularizations. We find that deep learning approaches for reliable communication require channel and code-specific intuitions and insights from information theory and coding theory.

Finally, we tackle the challenging problem of designing encoders and decoders jointly using RNNs in § V-B. The communication channel we consider is the AWGN 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 RNN based encoders and decoders to create novel codes that outperform known codes by 3 orders of magnitude in reliability. We also demonstrate several desirable properties of the codes: (a) generalization to larger block lengths, (b) composability with known codes, (c) adaptation to practical constraints.

These results bring to 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. In this article we have focused on a specific family of deep learning methods (namely, RNNs) which enlargen a specific family of coding theoretic methods (sequential coding and decoding). Considering a broader family of deep learning methods (eg: transformers) and a broader family of coding theoretic methods to enlargen (eg: algebraic codes such as Reed-Muller codes) is a promising area of research. We briefly discuss these directions along with further strengthening the family of RNNs in § VI; this concludes the paper.

II. Preliminary on convolutional codes and its variants

Among diverse families of coding schemes introduced in the literature, *sequential coding* schemes are particularly attractive for our goal; we aim to expand the space of efficient codes that practitioners can use, by searching for a good code over a larger space of functions for both encoders and decoders than traditionally done. Ideally, the search space needs to be large enough (in terms of the representation power of the class of functions we search over), while allowing for an efficient search (by exploiting and imposing structures that induce desirable properties). The structural similarities of *sequential codes* and *sequential neural networks* provide an ideal candidate for such a search space, that we investigate throughout this paper. This connection will be made mathematically precise in § III.

We use the term sequential codes to refer to *convolutional codes* and their variants like Turbo codes, i.e. codes (more specifically encoders) that apply a basic building block recursively over an input sequence. We explain them in detail in the following section. We use the term *sequential neural networks* to refer to a family of neural networks that apply a basic building block recursively over an input sequence. This includes Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), Gated Recurrent Units (GRU), and Long Short-Term Memory networks (LSTM), and we provide a brief summary in § III.

Our goal is to expand the space of sequential codes and discover powerful new codes, by building upon recent advances in sequential neural networks from the deep learning literature. This focus is based on the success of sequential codes, which

(a) are used extensively in mobile telephone standards including satellite communications, 3G, 4G, and LTE; (b) provably achieve performance close to the information theoretic limit (albeit with computationally inefficient decoding); and (c) have a natural recurrent structure that is aligned with an established family of deep models, namely sequential neural networks. In this section, we provide an overview of sequential codes.

A. Convolutional codes

A standard example is the *rate-1/2 Recursive Systematic Convolutional (RSC) code*. The encoder performs a forward pass recursively as shown in Figure 2 on a binary input sequence $\mathbf{b} = (b_1, \ldots, b_K) \in \{0,1\}^K$, which we call *message bits* or *information bits*, and computes a sequence of binary vector *states* $(s_1, \ldots, s_K) \in \{0,1\}^{2\times K}$ and a sequence of binary vector outputs $(c_1, \ldots, c_K) \in \{0,1\}^{2\times K}$, which we call *coded bits* or a *codeword*. At time k, the dynamic system is associated with a *state* represented by a two-dimensional binary vector $s_k = (s_{k1}, s_{k2})$ and takes as input a binary variable $b_k \in \{0,1\}$. The corresponding output is a two-dimensional binary vector $c_k = (c_{k1}, c_{k2}) = (b_k, b_k \oplus s_{k1}) \in \{0,1\}^2$, where $x \oplus y = |x - y|$. The state of the next cell is updated as $s_{k+1} = (b_k \oplus s_{k1} \oplus s_{k2}, s_{k1})$. Initial state is assumed to be 0, i.e., $s_1 = (0, 0)$.

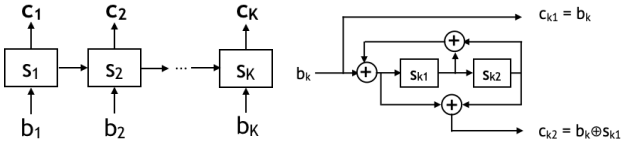


Fig. 2: (Left) A sequential encoder of a convolutional code is a recurrent network. (Right) An example of one cell for a rate-1/2 RSC code.

Fig. 3: The family of convolutional codes is a canonical sequential code, with a family of efficient decoders that is also optimal. We subsequently replace the decoder (Figure 7 in § IV-A) and also the encoder (Figure 11 in § IV-B) with trained neural networks, while maintaining the sequential structure for efficient training and decoding. We next investigate the potential benefits of sequential neural networks when the channel is not AWGN (Figure 16 in § V-A and Figure 18 in §V-B).

As illustrated in Figure 3, the coded bits $\mathbf{c} \in \{0,1\}^{2K}$ are mapped to *transmitted symbols* $\mathbf{x} = 2\mathbf{c} - 1 \in \{-1,1\}^{2\times K}$. This is known as binary phase shift keying (BPSK) modulation. Throughout this paper, we omit explicit description of the modulation and treat it as a part of the encoder (as done in Figure 3). The transmitted symbols are sent over a noisy channel, with the canonical one being the Additive White Gaussian Noise (AWGN) channel. The received real-valued vectors $\mathbf{y} = (y_1, \ldots, y_K) \in \mathbb{R}^{2\times K}$, which are called the *received symbols*, are $y_{ki} = x_{ki} + z_{ki}$ for all $k \in \{1, \ldots, K\}$ and $i \in \{1, 2\}$, where z_{ki}'s are i.i.d. Gaussian with zero mean and variance σ^2.

A decoder attempts to find the Maximum a Posteriori (MAP) estimate of the message bits $\mathbf{b}$, from received symbols $\mathbf{y}$. Due to the simple recurrent structure of the encoder, efficient iterative schemes are available for finding the MAP estimate exactly for any convolutional codes [4, 5]. There are two MAP decoders depending on the error criterion in evaluating the performance: bit error rate (BER) or block error rate (BLER).

The BLER evaluates the average fraction of the blocks that are erroneous, i.e. the decoder failed to recover the message $\mathbf{b}$ exactly (assuming many such length-K blocks have been transmitted). The matching optimal MAP estimator is $\hat{\mathbf{b}} = \arg\max_{\mathbf{b}\in\{0,1\}^K} \mathbb{P}(\mathbf{b}|\mathbf{y})$. One can find the MAP estimate in time linear in the block length K (and exponential in the size of the state, which is two in the above example of rate-1/2 RSC code), which is called *Viterbi algorithm*.

The BER evaluates the average fraction of message bits that are erroneous, i.e. the decoder failed to recover correctly in a block. The matching optimal MAP estimator for the k-th message bit is $\hat{b}_k = \arg\max_{b_k} \mathbb{P}(b_k|\mathbf{y})$, for all $k \in \{1, \cdots, K\}$. Again using a dynamic programming, the $\hat{b}_k$'s can be simultaneously computed in $O(K)$ time, which is called the *BCJR algorithm*.

In both cases, the optimal (ML) decoder is computationally very efficient (linear complexity) and crucially depends on the sequential structure of the encoder. This structure can be represented as a Hidden Markov Model (HMM). Both decoders are special cases of a general family of efficient methods to solve inference problems on HMMs using the principle of dynamic programming. These methods efficiently compute the exact posterior distributions in two passes through the network: the forward pass and the backward pass. A natural first step in determining if sequential neural networks are sufficient for achieving our goal is to to recover Viterbi and BCJR algorithms via neural network training. In § IV-A, we investigate if one can train a sequential neural network from samples, without explicitly specifying the underlying probabilistic model, and still recover the accuracy of the matching optimal decoders. At a high level, we want to prove by a constructive example that highly engineered dynamic programming (Viterbi algorithm) can be matched by a neural network which only has access to the samples from a dynamic system (the message sequence $\mathbf{b}$ and the noisy received sequence $\mathbf{y}$). The challenge lies in finding the right architecture and training with the right training examples.

B. Turbo codes

Turbo codes, which concatenate codewords from two convolutional encoders, are the first practical capacity-approaching codes [6]. A standard example is *the rate-1/3 Turbo code* that concatenates rate-1/2 RSC codes in parallel, as illustrated in Figure 4. The encoder maps $\mathbf{b} \in \{0,1\}^K$ to $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3) \in \{0,1\}^{3\times K}$. The coded bit streams $\mathbf{c}_1, \mathbf{c}_2$ are generated through

a single rate-1/2 RSC encoder that takes $\mathbf{b}$ as an input; $\mathbf{c}_1 = \mathbf{b} \in \{0,1\}^K$ denotes the systematic bits (information bits) and $\mathbf{c}_2$ denotes the (remainder of the) coded bits. The coded bit stream $\mathbf{c}_3$ is generated through a rate-1/2 RSC encoder that takes the interleaved (permuted) bit sequence $\pi(\mathbf{b}) \in \{0,1\}^K$ as an input. Systematic bit stream $\pi(\mathbf{b})$ is not included in the codeword as it is redundant. The interleaver plays a crucial role in introducing long range correlations, which contributes to the high reliability of Turbo codes.

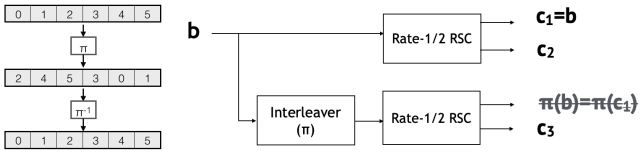


Fig. 4: (Left) Visualization of interleaver (π) and de-interleaver (π^{-1}). (Right) A rate-1/3 Turbo code which concatenates rate-1/2 RSC codewords in parallel.

The $3K$ output bits are modulated to $\mathbf{x} = 2\mathbf{c} - 1 \in \{-1,+1\}^{3\times K}$ and sent over a noisy channel, with the canonical one being the AWGN channel. Let $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3) \in \mathbb{R}^{3\times K}$, where $\mathbf{y}_i = \mathbf{x}_i + \mathbf{n}_i \in \mathbb{R}^K$ for $i \in \{1,2,3\}$, denotes the received real-valued vectors.

A decoder attempts to estimate $\mathbf{b}$ based on $\mathbf{y}$. Turbo codes accompany with corresponding iterative decoders which we call Turbo decoders. The posterior distribution is successively refined via the BCJR algorithm with interleavers and de-interleavers as shown in Figure 5. The Turbo decoder first computes the likelihood $\mathbf{L} = (L_1, \cdots, L_K)$ for $L_k = \log(\mathbb{P}(b_k = 1|(\mathbf{y}_1, \mathbf{y}_2))/\mathbb{P}(b_k = 0|(\mathbf{y}_1, \mathbf{y}_2)))$ based on the noisy rate-1/2 RSC codeword $(\mathbf{y}_1, \mathbf{y}_2)$ via BCJR algorithm. The Turbo decoder then estimates $\mathbb{P}(b_k|(\pi(\mathbf{y}_1), \mathbf{y}_3, \mathbf{L}))$ based on the noisy rate-1/2 RSC codeword $(\pi(\mathbf{y}_1), \mathbf{y}_3)$ with a prior given by $\mathbf{L}$ from the previous execution of a BCJR algorithm. The Turbo decoder iterates this process until convergence or a predefined number of iterations.

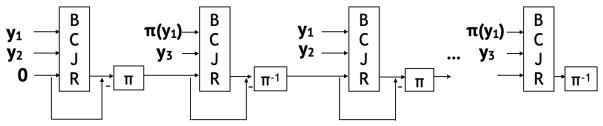
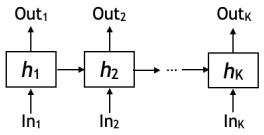


Fig. 5: Turbo decoder is an alternating recursion of two BCJR algorithms with interleaver (π) and de-interleaver (π^{-1}) in between.

III. Preliminary on Sequential Neural Networks

We focus on a family of neural architectures which recursively applies a functional mapping on the input sequence while maintaining a memory (referred to as a hidden state) to represent an output sequence. We refer to this class as *sequential neural networks*, as opposed to a more conventional recurrent neural networks, to reserve the term recurrent neural network (RNN) for a specific choice of the functional mappings as described below.

The sequential neural network family is a strictly larger class of functions than the family of encoders and decoders used in convolutional codes. In particular, one can manually choose the parameters of a sequential neural network to replicate any convolutional encoder and decoder.



$$h_k = f(h_{k-1}, \mathrm{In}_k)$$
$$\mathrm{Out}_k = g(h_k)$$

Fig. 6: A sequential neural network that maps an input sequence $(\mathrm{In}_1, \cdots, \mathrm{In}_K)$ to an output sequence $(\mathrm{Out}_1, \cdots, \mathrm{Out}_K)$.

As illustrated in Figure 6, a sequential neural network maps an input sequence $(\mathrm{In}_1, \cdots, \mathrm{In}_K)$ to an output sequence $(\mathrm{Out}_1, \cdots, \mathrm{Out}_K)$ in a sequential manner. For $k = 1, \cdots, K$, the hidden state h_k is updated as a parametric function f of the previous hidden state h_{k-1} and the k-th input In_k. The role of hidden state h_k is to capture information on past and current inputs $(\mathrm{In}_1, \cdots, \mathrm{In}_k)$ that is required to generate the output Out_k. The k-th output Out_k is a parametric function g of the hidden state h_k. Depending on which parametric functions f and g are used, the sequential neural network can be an RNN, an LSTM, or a GRU [7, 8, 9]. For example, an RNN maps $\mathbf{h}_k = \sigma_h(\mathbf{V}\mathbf{h}_{k-1} + \mathbf{U}\mathrm{In}_k + \mathbf{b}_h) \in \mathbb{R}^M$ where M denotes the dimension of hidden state $\mathbf{h}_k$, σ_h denotes an activation function, and $\mathbf{V} \in \mathbb{R}^{M\times M}, \mathbf{U} \in \mathbb{R}^{M\times|\mathrm{In}_k|}, \mathbf{b}_h \in \mathbb{R}^M$ are trainable parameters. Similarly, $\mathrm{Out}_k = \sigma_o(\mathbf{W}\mathbf{h}_k + \mathbf{b}_o)$, where σ_o denotes an activation function and $\mathbf{W} \in \mathbb{R}^{|\mathrm{Out}_k|\times M}, \mathbf{b}_o \in \mathbb{R}^{|\mathrm{Out}_k|}$ denote trainable parameters. Any convolutional encoder is a special case of an RNN. For example, let

$$\mathrm{In}_k = b_k, \mathrm{Out}_k = \begin{bmatrix} c_{k1} \\ c_{k2} \end{bmatrix}, \mathbf{h}_k = \begin{bmatrix} s_{k1} \\ s_{k2} \\ b_k \end{bmatrix}.$$

The rate-1/2 RSC code in Figure 2 can be represented as

$$\mathbf{h}_k = \sigma\left(\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\mathbf{h}_{k-1} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\mathrm{In}_k\right),$$
$$\mathrm{Out}_k = \sigma\left(\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}\mathbf{h}_k + \begin{bmatrix} 1 \\ 1 \end{bmatrix} b_k\right), \text{where}$$

the i-th element of $\sigma(x)$ is 1 if x_i is odd and 0 otherwise. The RNN class is the simplest class in sequential neural networks, as it is a special case of the GRU and LSTM classes, which are variants of RNNs. These variants of RNNs have been applied to great successes, especially in the domain of natural language processing. We explain some of the insights behind those recent advances briefly in § VI, where we discuss future research directions.

Regarding Viterbi and BCJR algorithms, it is proven in [10, 11] that there exist specific choices of the parameters and activation functions of a sequential neural network that recovers both of the decoders. This justifies our choice of sequential neural networks in decoding convolutional codes in § IV-A and in matching Turbo encoders in § IV-B.

IV. Recovering existing algorithms with deep learning

The standard AWGN channel is the canonical setting most widely studied in the literature, and improving upon the multiple decades of finite block-length coding theory is challenging. Instead, we first study the capability of deep learning in matching the known performances of existing codes.

The first step towards discovering a novel code is to fix the encoder as a convolutional code and recover the performance of the optimal decoder, by training a neural network decoder from samples (Figure 7). This is a necessary precursor to the more challenging tasks of designing new codes by training encoders and decoders jointly in § IV-B and shown in Figure 11. Without the capability to train an efficient decoder, there is little hope in searching over neural networks for good encoders.

A. Learning the BCJR algorithm for decoding convolutional codes

Viterbi algorithm is a special case of dynamic programming applied to hidden Markov models, discovered in [4]. No algorithm can achieve a smaller BLock Error Rate (BLER). A similar approach, known as BCJR algorithm introduced in [5], achieves the best Bit Error Rate (BER). An example of a rate-1/2 convolutional code is shown in Figure 2.

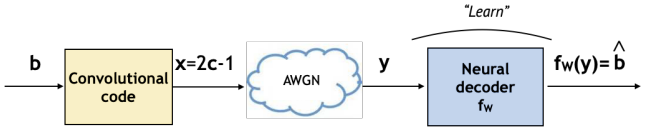


Fig. 7: The first step in designing new codes is to showcase that efficient and reliable decoders can be trained from samples. This is achieved by replacing the optimal decoder with a neural network, training the parameters $\mathbf{W}$ using training samples, and matching the optimal performance.

We first consider a simpler task of training a neural network decoder from N labelled samples of the form $\{(\mathbf{y}^{(i)}, \mathbb{P}(b_k|\mathbf{y}^{(i)}))\}_{i=1}^{N}$ for each $k \in \{1, \ldots, K\}$. The transmitted symbols $\mathbf{y}^{(i)}$ are the input to the neural network (generated by drawing a random codeword and simulating the channel), and the label $\mathbb{P}(b_k|\mathbf{y}^{(i)})$ (generated by running the BCJR algorithm) is the target we want to predict. A more challenging task of training without the help of the BCJR algorithm is a challenging but necessary step, and is addressed in the end of this section.

Our aim is to learn to mimic a dynamic programming (i.e. the BCJR algorithm) from examples. Solving such well-defined mathematical problems with deep learning has been recently studied in the machine learning community. This includes clustering under stochastic block models [12], traveling salesman problems [13], graph matching [14], and other combinatorial problems [15]. There are two main challenges in such endeavor. First, we want the end product to be an *algorithm*. This requires *generalization beyond the examples shown in the training*. It has been reported in [15] that neural network based solutions such as those in [13] do not generalize, especially in the size of the problem. This is especially concerning as training over a large dimensional examples is prohibitably slow. A key idea to overcome this challenge is sequential neural networks, as demonstrated in [15]. The recurrent structure is critical in achieving generalization to larger problems. We utilize this strength of sequential neural networks, on top of the fact that they match the structure of the encoder (i.e. convolutional codes) as well as the structure of the optimal decoder (i.e. BCJR decoder), to achieve strong generalization over the size of the problem (i.e. block length).

The second challenge is in choosing the *right training examples*. In the above mathematically defined problems, we have the freedom to generate (a possibly infinite number of) training examples. However, selecting the right training examples is critical in efficient training. [15] demonstrates that for the traveling salesman problem, choosing examples that are difficult enough allows the training to converge much faster. In our communication setting, we propose a novel scheme for choosing such training examples by fully utilizing the mathematical description of our problem. This turns out to be critical in achieving the accuracy we want in Figure 9 (left panel). Further, this principle turns out to be universally crucial for all scenarios we consider including joint encoder/decoder training (§ IV-B) and training feedback codes (§ V-B).

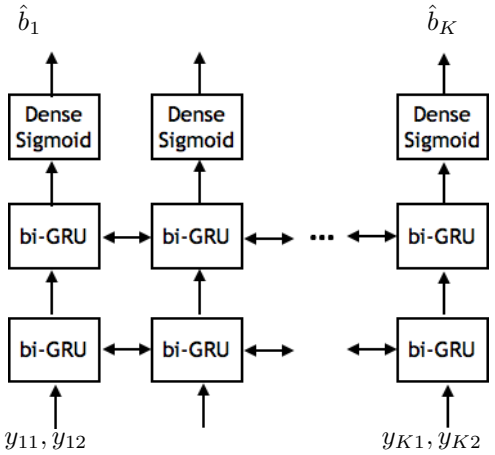
Architecture. To recover the performance of the BCJR algorithm with a trained neural network, we propose using bidirectional Gated Recurrent Units (bi-GRU). Let $\mathbf{y} = ((y_{11}, y_{12}), \ldots, (y_{K1}, y_{K2})) \in \mathbb{R}^{2\times K}$ denote the received symbols (as defined in § II-A) and let $f_{\mathbf{W}}(\mathbf{y}) = \hat{\mathbf{b}} \in \mathbb{R}^K$ denote the output of the neural network with a parameter $\mathbf{W}$. The following loss function requires an oracle access to the optimal BCJR algorithm in order to achieve the desired performance, and solves for

$$\text{minimize}_{\mathbf{W}} \ \mathbb{E}\Big[\sum_{k=1}^{K} \big(\hat{b}_k - \mathbb{P}(b_k = 1|\mathbf{y})\big)^2 \Big], \qquad (1)$$

where the expectation is over the samples used in training, and the conditional probability is evaluated using the BCJR algorithm. This is a standard quadratic loss for a regression problem.

Results. Consider decoding convolutional codes under the Additive White Gaussian Noise (AWGN) channel with a specific Signal-to-Noise Ratio (SNR) defined as $-10\log_{10}\sigma^2$ where σ^2 is the variance of the Gaussian noise in the channel (assuming the transmitted sequence $\mathbf{x}$ satisfis $\mathbb{E}[\mathbf{x}^2] = 1$). A standard machine learning approach is to train on example sequences from the same channel and the same SNR as it will be tested on. This fails to achieve the BCJR performance as shown in Figure 9 (top panel). Perhaps surprisingly, there exists an empirically optimal training SNR (0dB in the rate-1/2 example below) that is different from the testing SNRs. Further, this optimal training SNR primarily only depends on the rate of the code.

We give a rule for choosing such training examples, borrowing information theoretic insight. Intuitively, we aim to train on examples that are just difficult enough (close to the



Layer	Output dimension (# of units, dimension)
Input	$(K, 2)$
1st bi-GRU	$(K, 400)$
1st Batch Normalization	$(K, 400)$
2nd bi-GRU	$(K, 400)$
2nd Batch Normalization	$(K, 400)$
Dense (sigmoid)	$(K, 1)$

Fig. 8: Neural decoder architecture in [16] for the rate 1/2-RSC code in Figure 3.

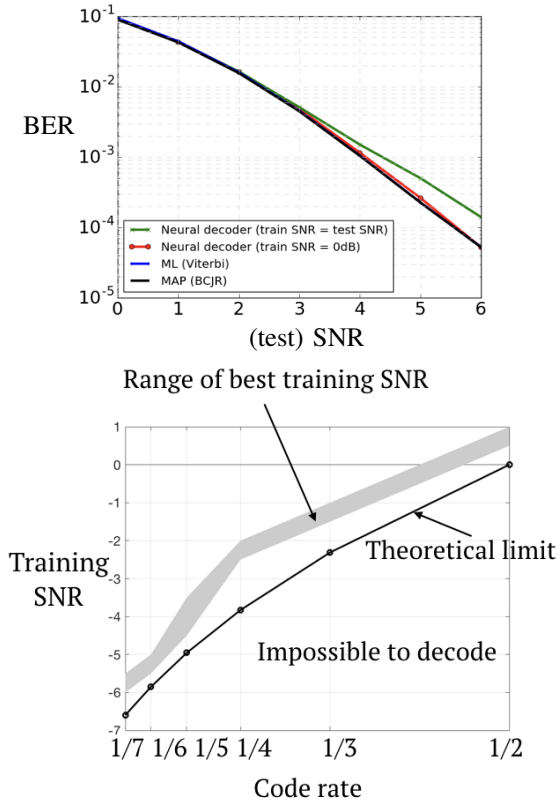


Fig. 9: Neural decoder performance for the rate 1/2-RSC code in Figure 2. (Top) Neural decoder trained at a mismatched 0dB SNR improves upon a standard matched training, when trained and tested with block length $K = 100$. (Bottom) The region of optimal training SNR is shown in grey as a function of the rate of the code. It is close to an information theoretic prediction shown in black line.

decision boundaries) so that no training time is wasted (on easy examples). Typically, this requires adaptive schemes, as the decision boundary is not known. However, as the model is mathematically precise in the channel decoding, we do not need adaptive schemes to figure out where the decision boundary is. Instead, we harness information theoretic insights to prescribe an analytically chosen training SNR level (0dB in this example with rate 1/2 codes) as follows. We parametrize the noise we add in the training examples by their SNR. To find the right *training SNR*, we need to know the (average) distance from a codeword to the decision boundary, which changes for different encoding schemes. To get a universal rule of thumb, we consider the ideal codewords with maximum distance between neighboring codes (such as the Gaussian code book). For a given code rate r, this ideal distance to the boundary can be translated into

$$\text{SNR}_{\text{train}} \quad = \quad 10 \log_{10}(2^{2r} - 1) \, , \tag{2}$$

and is shown in black line named "Theoretical limit" in Figure 9 (bottom panel). This is derived from the sphere packing bound (and also Shannon capacity of the Gaussian channel). As typical codes have smaller inter-codeword distances than the ideal code, the empirical best choice of training SNR is slightly larger (but close to the theoretical prediction), shown in grey.[1] Similar efforts towards understanding optimal choices of training data for training decoders have been taken in [17], and active learning approaches were explored in [18].

One would claim that the BCJR algorithm is recovered only if a neural decoder trained at a small block length can be applied to a larger block lengths. Figure 10 shows how such generalization in block length is achieved by the proposed approach. Such generalization property is one of the main challenges in recovering existing algorithms with deep learning.

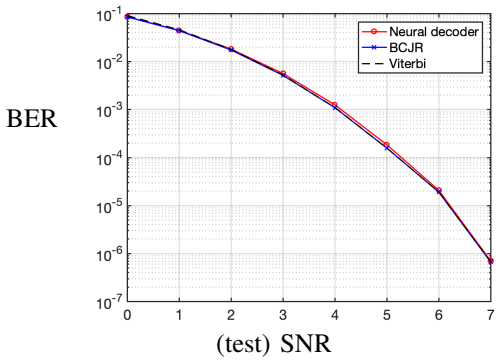


Fig. 10: The neural decoder trained on $K = 100$ sequences generalizes to match the BCJR decoder when tested on block length $K = 10,000$ sequences.

Related work. Building upon the above neural decoder, one

[1]The codes from [16] are available at https://github.com/deepcomm/RNNViterbi

can immediately decode Turbo codes as shown in [16]. However, this requires access to the BCJR algorithm. The question of whether this is necessary was resolved in [19], where a standard cross entropy loss of $\ell(\hat{b}_k, b_k) = -b_k \log(\hat{b}_k) - (1-b_k)\log(1-\hat{b}_k)$ was used (with no oracle access to the BCJR) on a training data of the form $\{(\mathbf{b}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$. This is a standard loss used in training binary classifiers. To achieve the performance of the Turbo decoder without access to the BCJR decoder, an additional technique is necessary: using multi-dimensional vectors to encode the posterior beliefs from one iteration to the next. The standard Turbo decoder encodes its posterior belief as a scalar valued log-likelihood, when going from one iteration to the next. Using multi-dimensional posterior beliefs allows the neural network to search over a larger class of functions that can potentially encode more information between iterations, thus achieving the performance of a Turbo decoder without the help of BCJR codes.

Following [16], [20] compares the capability of various neural networks in achieving Viterbi performance on convolutional codes and proposes a new training method for learning Viterbi decoders for convolutional codes with a long memory. [21] studies non-AWGN channels (e.g., time varying channels or channels with memory) and explores neural decoders that match Viterbi performance.

First attempts to design neural decoders used general purpose neural networks. With no structural insights to guide the training process, typical approaches in this direction were limited to small block lengths. [22] designs a neural network decoder that can closely match the optimal performance of MAP decoder for the Polar code of a fixed size of 16 bits. In a series of work, [23, 24, 25, 26] introduce trainable weights to the popular belief propagation decoder and trains those parameters via supervised training with examples of pairs of noisy codewords and the true message bit sequence. Decoders for several High Density Parity Code (HDPC) codes (e.g., BCH(127,106), BCH(63,36), BCH(63,45)) are considered in [23], where belief propagation decoders are known to have inferior performances. Near MAP performances were achieved on HDPC codes in the subsequent work [27]. A similar idea of weighted belief propagation is used in [28] for (128,64) and (256,128) polar codes, and CRC-polar concatenated codes in [29]. [26] demonstrates near MAP performances on the $(32, 16)$ Reed–Muller code. [30] uses reinforcement learning to find the best decoder for a given scenario, achieving near optimal performances on BCH and Reed-Muller codes.

General purpose neural network decoder architectures have also been proposed, without exploiting any specific structure of the encoder (other than linearity). In [31] the idea of syndrome decoding is used to avoid overftting to the codewords shown in the training. Perhaps surprisingly, RNNs are shown to achieve the best performance in this unstructured setting also (when trained and tested with BCH codes).

B. Learning Turbo codes

Given the success in learning decoders, a natural question is whether we can learn both the encoder and the decoder via deep learning and recover the performance of existing codes. We model both the encoder and the decoder as neural networks, as illustrated in Figure 11. When trained with the right neural architectures and the right training methodology, the pair of encoder and decoder achieves the reliability of Turbo codes under the AWGN channel.

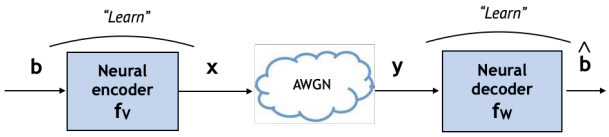


Fig. 11: Both the encoder and the decoder are modeled as neural networks, while maintaining the sequential structure, and the parameters $\mathbf{V}$ and $\mathbf{W}$ are trained using training samples for AWGN channels.

For the architectures, sequential neural networks, such as an Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs), are natural choices for both the encoder and the decoder, (as explained in § II and § III). However, the accuracy is significantly worse when compared to Turbo codes (see the performance of CNN-AE in Figure 14). We propose a novel architecture inspired by Turbo codes, called *Turbo Autoencoder (TurboAE)*, which harmoniously combines the CNN architecture with the interleaver of Turbo codes for the encoder [32][2]. Correspondingly, we model the decoder as a layered CNNs with an alternating application of the interleaver and the deinterleaver, inspired by the Turbo decoder.

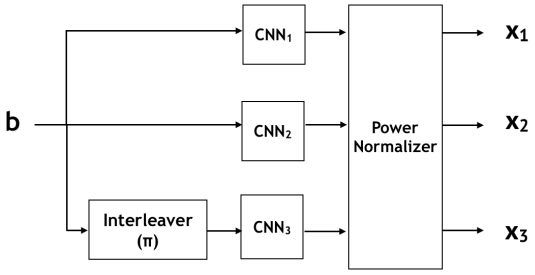
Training two neural networks jointly is challenging. Borrowing ideas from alternating gradient methods used in training Generative Adversarial Networks (GAN) [33, 34], we propose the following training strategies: first, we train the encoder and the decoder via two alternating gradient descents. We update the decoder for a fixed encoder for a larger number of iterations than the number of iterations used when updating the encoder. This prevents the encoder from chasing after a mismatched decoder. Such two-timescale gradient methods are successful in solving a certain class of minimax optimization problems, like those in training GANs [35]. Secondly, the SNR of the training examples are carefully chosen, similarly as in § IV-A. Lastly, it is critical to use a order of a magnitude larger batch size (compared to when training only the decoder). Such a benefit of a larger batch size has been observed empirically in other domains where alternating gradient methods are used (e.g. in training GANs [36]).

Setup. Concretely, we consider learning a *rate-1/3* code for $K = 100$ information bits, i.e., the encoder maps $\mathbf{b} \in \{0,1\}^{100}$ to a codeword $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \in \mathbb{R}^{3\times 100}$, and the decoder maps $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3) \in \mathbb{R}^{3\times 100}$ to $\hat{\mathbf{b}} \in \{0,1\}^{100}$, where $\mathbf{y}_i = \mathbf{x}_i + \mathbf{z}_i$ for i.i.d. Gaussian noise sequence $\mathbf{z}_i$ for $i = 1, 2, 3$. We compare against existing codes, including Turbo, polar, and LDPC codes [6, 37, 38].

Architecture. TurboAE consists of a channel encoder and a decoder. Each is modeled as a neural network. The design is

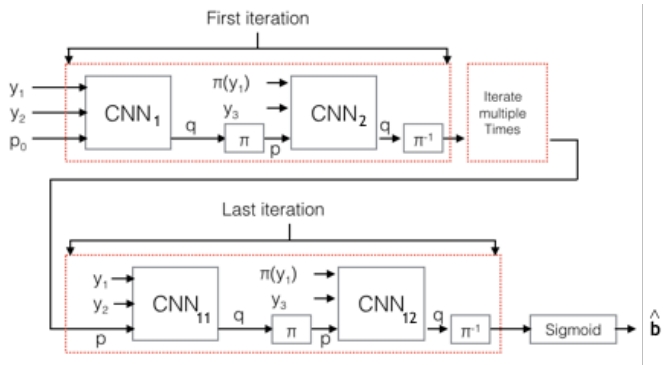
[2]Implementation of [32] is available at `https://github.com/yihanjiang/Turboae`.

inspired by rate-1/3 Turbo encoders and Turbo decoders. As shown in Figure 12, the neural encoder of TurboAE combines the interleaver and convolutional neural networks (CNNs); it consists of three trainable CNN encoding blocks followed by a power normalization layer. Input to the first two CNN encoding blocks is the original bit sequence, while input to the last CNN block is an interleaved bit sequence. The detailed architecture of each CNN block is shown in the table in Figure 12, where the filter size of each CNN layer is five.



Encoder	Layer	Output dimension
Block CNN$_i$	Input	(K,1)
$(i = 1, 2, 3)$	CNN + ReLU	(K,100)
	CNN + ReLU	(K,100)
	Linear + ELU	(K,1)

Fig. 12: The encoder architecture in Turbo Autoencoder for a rate-1/3 code.



Decoder	Layer	Output dimension
Block CNN$_i$	Input	(K,7)
$(i = 1, \cdots, 12)$	CNN + ReLU	(K,100)
	CNN + ReLU	(K,100)
	CNN + ReLU	(K,100)
	CNN + ReLU	(K,100)
	CNN + ReLU	(K,100)
	Linear + ELU	(K,1) for $i = 1, \cdots, 11$
		(K,5) for $i = 12$

Fig. 13: The decoder architecture in Turbo Autoencoder for a rate-1/3 code.

The neural decoder of TurboAE consists of several layers of convolutional neural networks with interleavers and de-interleavers in between, as shown in Figure 13. Let $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3 \in \mathbb{R}^K$ denote noisy versions of $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \in \mathbb{R}^K$, respectively. The decoder generates an estimate $\mathbf{b} \in \{0,1\}^K$ based on $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3$ via multiple iterations of CNNs. Two types of decoders are alternatively applied, CNN$_i$ and CNN$_{i+1}$, with an interleaver and de-interleaver in between, for $i \in \{1, 3, 5, 7, 9, 11\}$. The first decoder, CNN$_i$, takes received signals $\mathbf{y}_1$, $\mathbf{y}_2$ and the prior $\mathbf{p}$ of size (K, f). Standard Turbo decoders use $f = 1$ as the prior belief for each information bit, which represents a scalar valued log-likelihood ratio. However, we let $f = 5$ as we observe that $f > 1$ significantly improves the accuracy of the final encoder-decoder pair. Such over-parametrization is known to improve the loss landscape in training and achieve a better generalization, in the machine learning literature [39, 40, 41]. Conceptually, similar to Turbo decoders, the CNN$_i$ generates the posterior belief $\mathbf{q}$ of size (K, f) given the prior belief p and $\mathbf{y}_1, \mathbf{y}_2$ as inputs.

The second decoder, CNN$_{i+1}$, then takes the interleaved signal $\pi(\mathbf{y}_1)$, received symbols $\mathbf{y}_3$, and interleaved prior $\mathbf{p}$ and generates the posterior belief $\mathbf{q}$, also of size (K, f). The first iteration takes $\mathbf{p}_0 = 0$ (of size (K, f)) as a prior, and the final CNN layer CNN$_{12}$ generates a posterior of size $(K, 1)$, which is passed through a sigmoid function to generate $\hat{\mathbf{b}} = \text{round}(\text{sigmoid}(\mathbf{q}))$.

As the CNNs have enough representation power to mimic convolutional codes and BCJR decoders (see §III), The proposed architecture can in principle mimic Turbo encoders and decoders. However, training TurboAE architectures from data requires the following additional techniques, to ensure that we learn to exploit the full potential of the long-range correlations provided by the interleavers.

- **Alternating the training of the encoder and the decoder.** Let $\mathbf{V}$ and $\mathbf{W}$ denote the parameters in the encoder and the decoder neural networks, respectively. Stochastic gradient descent is used to train $\mathbf{V}$ and $\mathbf{W}$, which solves

$$\text{minimize}_{\mathbf{V},\mathbf{W}} \ \mathbb{E}\Big[\sum_{k=1}^{K} \ell(b_k, \text{sigmoid}(q_k)) \Big] ,$$

  where we use a cross-entropy loss $\ell(b_k, \text{sigmoid}(q_k)) = -\{b_k \log(\text{sigmoid}(q_k)) + (1 - b_k) \log(1 - \text{sigmoid}(q_k))\}$, and the expectation is over the samples used in the training. As opposed to updating $\mathbf{V}$ and $\mathbf{W}$ simultaneously, as in [42], we alternate the training of $\mathbf{V}$ and $\mathbf{W}$ to prevent converging to a bad local optimum [43] [44]. The decoder and the encoder are trained asymmetrically, which is critical in achieving an improved accuracy. The decoder $\mathbf{W}$ is updated for a larger number of iterations than the encoder $\mathbf{V}$ is (e.g., $\mathbf{W}$ is updated with 500 examples, $\mathbf{V}$ with 100 examples).
- **Different training noise levels for the encoder and decoder.** It has been shown empirically that it is best to train the decoder at a lower training SNR than the SNR it will be tested on SNR [16, 44]. However, in joint encoder-decoder training setting, as the encoder is evolving over the training process (and corresponding decision boundaries between codewords also shifting), [32] trains the decoder at various SNRs (random selection from -1.5 to 2 dB). We refer to §IV-A for the insight in how to choose the training SNRs.
- **Large batch size.** A large batch size is critical in our training. TurboAE trained with a mini-batch size less than

500 achieves a noticeably worse accuracy.

Result. Figure 14 shows a BER vs. SNR graph for TurboAE, with widely used baseline codes including Turbo, polar, LDPC, and tail-biting convolutional codes (TBCCs) generated via Vienna 5G simulator [45] [46] for block length 100 and code rate 1/3. CNN-AE is a trained CNN-based encoder-decoder architecture that does not use an interleaver. This shows that simply replacing the encoder and decoder with general-purpose neural networks does not achieve a high accuracy.

TurboAE has two versions. TurboAE-continuous, reviewed in this tutorial, enforces a soft power constraint and allows real-valued codewords. TurboAE-binary, on the other hand, enforces the codewords to be binary. Both versions of TurboAE are shown to perform comparable to Turbo codes at a low SNR, while at a high SNR (over 2 dB with BER $< 10^{-5}$), performance is worse than LDPC and polar codes.

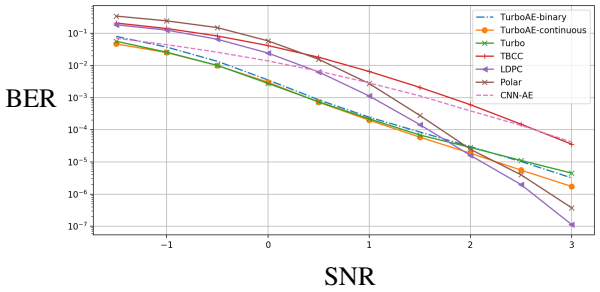


Fig. 14: TurboAE recovers the reliability of modern codes for moderate block length (100 information bits, rate 1/3) in low and mid-range of SNRs

The BERs of TurboAE, Turbo code, and CNN-AE vs. the block length are shown in Figure 15 (top panel), where the SNR of the AWGN channel is fixed at 2dB. As block length increases, improved reliability can be achieved, which we refer to as the *block length gain*. Modern codes that have a long-range memory, such as Turbo codes, achieve this block length gain, while convolutional codes do not. TurboAE also exploits a long-term memory by the embedded interleaver and achieves the block length gain, while a direct application of well-known neural architectures (e.g., CNN-AE) tends to exploit only a short-range memory and does not achieve the block length gain. In Figure 15 (bottom panel), the reliability of TurboAE with and without a random interleaver is shown, which demonstrates that the random interleaver is critical in achieving the accuracy of Turbo codes.

Related work. [42] introduces a new framework for end-to-end joint training of the encoder and the decoder. It is shown that thus trained encoder and decoder can recover the accuracy of a Hamming code for a short information block length ($K = 4, n = 7$). However, these general purpose neural networks do not scale to larger block lengths, as no structure is imposed. Addressing this scaling challenge, [32] imposes the sequential structure with an interleaver, recovering the accuracy of turbo codes on significantly longer block lengths.

As opposed to fully training both the encoder and decoder end-to-end, [47] applies deep learning to the polar code

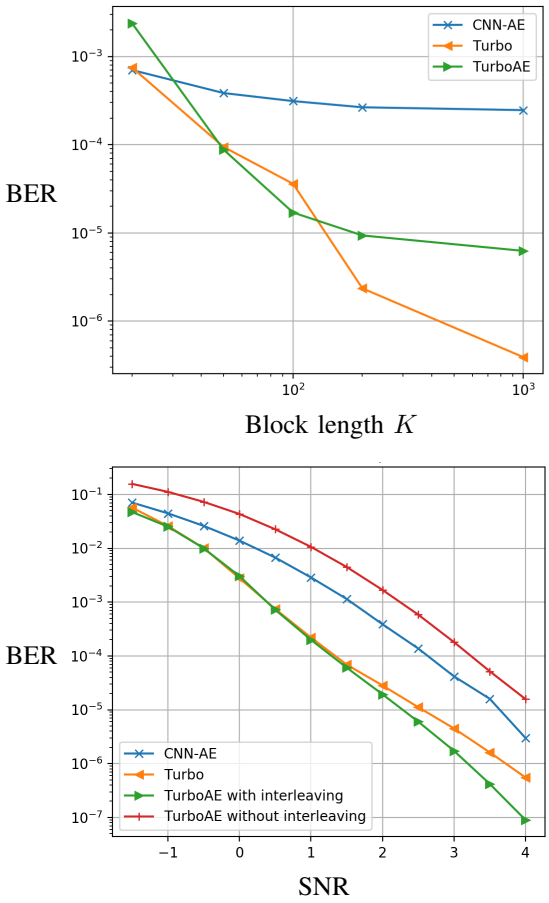


Fig. 15: (Top) The error probability of TurboAE (at SNR 2dB) decreases as block length K increases. (Bottom) The interleaver is critical in achieving the reliability of TurboAE.

construction (encoder side) tailored to the traditional belief propagation decoder. [48] uses reinforcement learning and genetic algorithm-based approaches to construct linear codes and polar codes. Similarly, [49] shows that polar codes constructed based on the genetic algorithm for (plain) successive cancellation list decoding achieve a reliability comparable to polar codes with CRC-aided SCL decoding.

V. New results on non-AWGN channels

On several channels that are not AWGN or even point-to-point, we often lack the technical tools to design good codes. Neural networks provide a promising alternative approach. On non-AWGN channels, we demonstrate that trained neural decoders are more robust to mismatched channel models than the standard Turbo decoders in § V-A. Further, when we can obtain training samples from such non-standard channels, neural decoders can adapt to new channels without any knowledge of the mathematical descriptions of the channel.

We next consider in § V-B the Gaussian noise channel with output feedback, whose study was initiated by Shannon; feedback is known theoretically to improve reliability of communication, but linear coding methods to incorporate the feedback have provably inferior performance compared to non-linear codes [50]. The space of non-linear codes is challenging

to explore with human ingenuity. We break this gridlock by training neural encoders and decoders from samples.

A. Neural decoder for non-AWGN channels

In practical wireless communications, the channels do not exactly match the mathematical models. Neural decoders are shown to have superior robustness and adaptivity relative to existing decoders when the channel deviates from the AWGN channels. *Robustness* refers to the ability of a decoder trained for a particular channel (e.g., AWGN) to work well, *without re-training*, when the test channel deviates from the training channel. *Adaptivity* refers to the decoder's ability to adapt to various non-AWGN channel models, by training with examples from those channels.

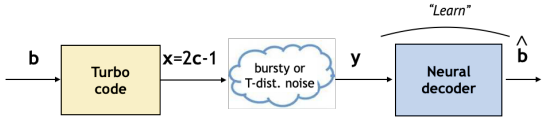
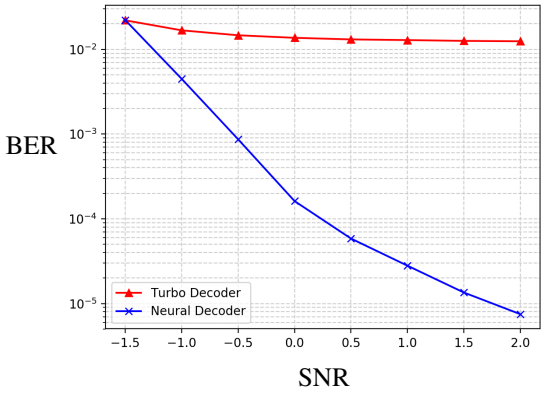


Fig. 16: Trained neural decoders exhibit superior robustness and adaptivity.

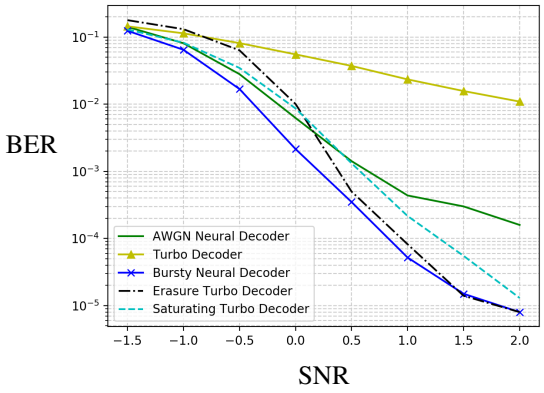
Figure 17 (top) shows that without any re-training, neural decoders trained on AWGN is robust against a channel that adds an i.i.d. noise according to the T-distribution (we use standard Turbo encoder in this example). Further, investigations reveal that Turbo decoders are (sometimes mistakenly) over-confident about some received symbols, and the propagation of such confident mis-information hurts the performance significantly. On the other hand, neural decoders are inherently conservative, making it robust against changes in the channel.

In Figure 17 (bottom), we demonstrate adaptivity results on bursty noise channels, where, with a small probability, a high-variance noise is added to the received symbol. This channel model captures how radar signals (which are bursty) can create an interference for wireless systems. For bursty noise channels, heuristic decoders are used in practice; the received symbols with high magnitudes are first thresholded before passing through the Turbo decoder. The performance of two such heuristics are shown in the figure as "Saturating Turbo Decoder" (which shrinks large magnitude signals) and "Erasure Turbo Decoder" (which sets large magnitude signals to zero). The performance of the "Turbo Decoder" tailored for the AWGN channel is shown as a reference. A neural decoder trained on AWGN channels is robust as expected (shown as "AWGN Neural Decoder"). The figure shows that the "Bursty Neural Decoder" trained on the bursty channel adapts to the new channel and achieves the best performance.

Related work. Traditional decoding algorithms, such Viterbi and BCJR algorithms, require the knowledge on the channel model (i.e., $\mathbb{P}(y|x)$ where x and y denote transmitted and received symbols, respectively). For i.i.d. AWGN channels, the noise variance fully characterizes the channel model, and it is assumed to be readily available at the receiver. On the other hand, for non-AWGN channels, the receiver often do not have the full knowledge on the channel model. In addition, decoding algorithm becomes more complicated



(a) Robustness result on T-Distribution channels



(b) Adaptivity result on bursty noise channels

Fig. 17: Neural network based decoders for Turbo codes are more robust when tested on T-distribution channels (top) and adapt to bursty noise channels to outperform heuristics used in practice (bottom).

as channel models become complicated; for example, the complexity of Viterbi algorithm increases exponentially in the memory of a channel (for channels with memory). To address these challenges, [21] proposes ViterbiNet which maintains the structure of Viterbi algorithm while the component of Viterbi algorithm that depends on the channel knowledge (computation of likelihood for each symbol) is replaced by a trainable neural network. It is shown that ViterbiNet is robust to the uncertainty in channel models. For example, ViterbiNet trained online based on recent decoding decisions closely achieves the accuracy of the optimal Viterbi algorithm that knows the precise (time-varying) channel model. [51, 52] also investigates deep learning based decoders for non-AWGN channels.

B. Learning codes for channels with output feedback

The advantage of deep learning for communication systems is that training can be done for channels for which analytically designing reliable codes is challenging. One such example is the output feedback channel in Figure 18, where the received symbols are sent back to the transmitter. Due to the natural sequential nature of this channel, sequential neural networks provide a promising direction for discovering a new code.

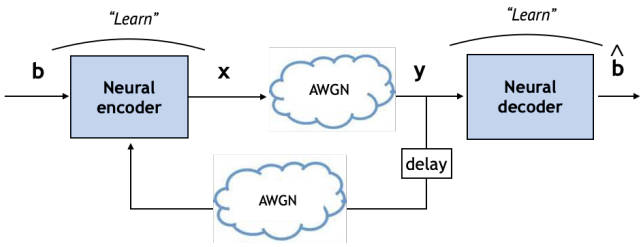


Fig. 18: End-to-end deep learning based communication system for channels with output feedback: both the encoder and decoder are modelled as neural networks and the parameters $\mathbf{V}$ and $\mathbf{W}$ are trained from samples.

In [53], novel codes based on RNNs are shown to operate significantly better (several orders of magnitude improvement in accuracy for certain regimes) than the state of the art on the AWGN channel with (noisy) output feedback.[3]

Setup. Modern wireless communication involves feedback from receiver to transmitter. The feedback can be in several forms (e.g., the received value itself or the acknowledgment of reception). Among several channel models, we consider channels with output feedback proposed by Shannon. As illustrated in Figure 19, both channels from/to the transmitter to/from the receiver are modeled as AWGN channels. At time i, the decoder receives $y_i = x_i + z_i$, where $z_i \sim \mathcal{N}(0, \sigma^2)$, and transmits y_i back to the encoder with a unit-step delay. At time i, the encoder receives $\tilde{y}_{i-1} = y_{i-1} + w_{i-1}$ for $w_{i-1} \sim \mathcal{N}(0, \sigma_F^2)$, which is an AWGN-added version of y_{i-1}. Formally, an *encoder* is a function that sequentially maps the information bit sequence $\mathbf{b} \in \{0,1\}^K$ and the feedback signals $\tilde{y}_1^{i-1} = (\tilde{y}_1, \cdots, \tilde{y}_{i-1})$ received thus far to a transmit signal x_i. The encoder is a mapping $f_i\colon\ (\mathbf{b}, \tilde{y}_1^{i-1}) \mapsto x_i \in \mathbb{R}, \quad i \in \{1, \cdots, n\}$. Without loss of generality, the average power of $\mathbf{x}$ is constrained to one, 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 and the randomness in the noisy feedback signals $(\tilde{y}_1, \cdots, \tilde{y}_n)$. A *decoder* is a function that maps the received sequence $\mathbf{y} = (y_1, \cdots, y_n)$ into estimated information bit sequence $g\colon\ \mathbf{y} \mapsto \hat{\mathbf{b}} \in \{0,1\}^K$.

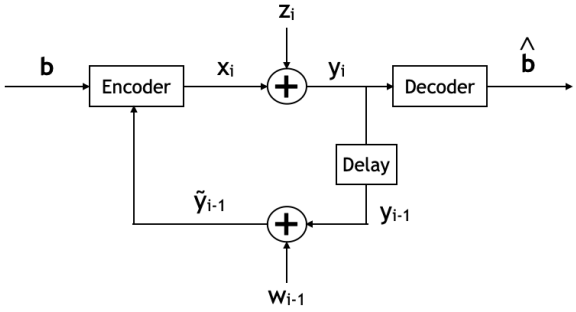


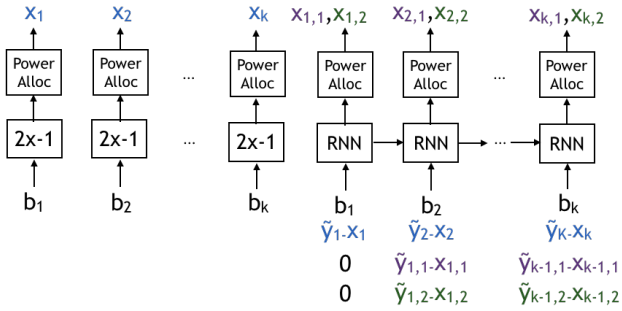
Fig. 19: An AWGN channel with noisy output feedback

[3]Implementation of [53] is available at https://github.com/hyejikim1/Deepcode,https://github.com/yihanjiang/feedback_code.

[54] shows that the capacity of the channel remains unchanged in the presence of output feedback. However, accuracy in the finite block length regime can in theory increase significantly, as demonstrated by the celebrated result of Schalkwijk and Kailath (S-K scheme) [55]. In practice, however, the S-K scheme is sensitive to the finite machine precision and noise in the feedback [55, 56]. We demonstrate that a trained pair of an encoder and a decoder outperforms the S-K scheme in accuracy (for $k = 50$ and code rate $1/3$ $(n = 3k)$) for the channels with output feedback.

...

Architecture. Both the encoder and the decoder are modeled as recurrent neural networks, as illustrated in Figures 20 and 21. This framework and the corresponding newly discovered code via deep learning is referred to as *DeepCode*. The rate 1/3 encoder operates in two phases. In the first phase, the encoder generates an uncoded transmission sequence of length K in $\mathbb{R}^K$, which is an output of a non-uniform and learnable power allocation applied to the information bit sequence. In the second phase, the encoder generates a coded transmission sequence of length $2K$ via an RNN followed by a learnable power allocation block. Each j-th RNN cell generates a pair of coded symbols $(x_{j,1}, x_{j,2}) \in \mathbb{R}^2$ as a function of the hidden state of the RNN, input information bit b_j, $\tilde{y}_j - x_j$ (the estimated noise added to the j-th transmission in phase 1) and $(\tilde{y}_{j-1,1} - x_{j-1,1}, \tilde{y}_{j-1,2} - x_{j-1,2})$ (the estimated noise added to the $(j-1)$-th transmission in phase 2).

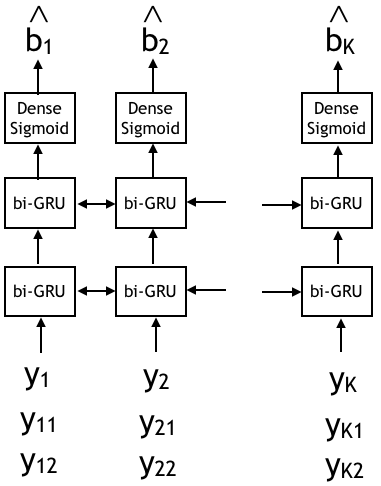


Layer	Output dimension
Input	(K, 4)
RNN (tanh)	(K, 50)
Dense (sigmoid)	(K, 2)
Normalization	(K, 2)

Fig. 20: The encoder architecture of the rate-1/3 DeepCode in [53]

The decoder architecture is shown in Figure 21. Let $\mathbf{y} = (y_1, \cdots, y_K, y_{1,1}, y_{1,2}, \cdots, y_{K,1}, y_{K,2}) \in \mathbb{R}^{3K}$ denote the received sequence, i.e., $y_j = x_j + z_j$, $y_{j,1} = x_{j,1} + z_{j,1}$, and $y_{j,2} = x_{j,2} + z_{j,2}$, where $z_j, z_{j,1}, z_{j,2}$ denote the Gaussian noise added in the AWGN channel for $j = 1, \cdots, K$. The decoder maps $\mathbf{y}$ to $\hat{\mathbf{b}} \in \{0,1\}^K$ via two-layered bidirectional GRUs, where the input to the k-th first-layer GRU cell is a tuple of three received symbols, $(y_k, y_{k,1}, y_{k,2})$.

The encoder and the decoder are trained jointly via back propagation through time (on the entire input message bit sequence) to minimize the binary cross-entropy loss function.



Layer	Output dimension
Input	(K, 3)
bi-GRU	(K, 100)
Batch Normalization	(K, 100)
bi-GRU	(K, 100)
Batch Normalization	(K, 100)
Dense (sigmoid)	(K, 1)

Fig. 21: The decoder architecture of the rate-1/3 DeepCode in [53]

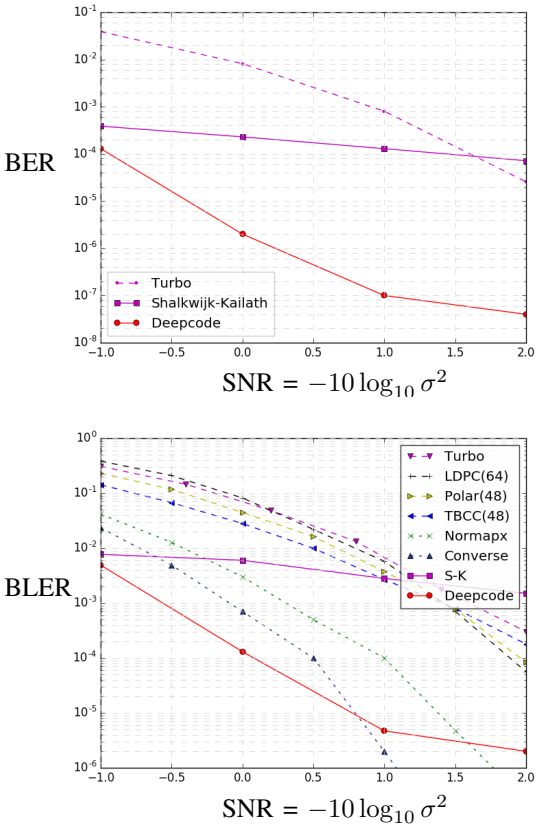


Fig. 22: Deepcode outperforms the baseline of S-K and Turbo code by several orders of magnitude in BER, on block-length 50 and noiseless feedback in BER (top) and BLER (bottom). DeepCode also outperforms all state-of-the art codes (that do not use feedback) in BLER (bottom).

Result. The BER and BLER of DeepCode and several baselines are shown in Figure 22 for channels with noiseless feedback (i.e., $\sigma_F^2 = 0$) for input block length $K = 50$ and code rate $1/3$. The S-K scheme uses a practical choice of MATLAB implementation with a precision of 64 bits to represent floating-point numbers. Since the scheme is very sensitive to finite numerical precision, numerical errors dominate the performance of the S-K scheme, as shown in Figure 22. At a moderate SNR of 2 dB, DeepCode outperforms the S-K scheme by three orders of magnitude in BER.

In Figure 22 (bottom), the BLER of DeepCode is shown together with the state-of-the-art polar, LDPC, and convolutional codes (from [57]) that do not use feedback. Notably, DeepCode significantly improves over all state-of-the-art codes of similar block-lengths and the same rate. In addition, the theoretical estimate of the best code (with no efficient decoding schemes) for channels without feedback is shown as a reference, which lies between an approximate achievable BLER (labelled Normapx) and a converse to the BLER (labelled Converse) from [58, 59].

Related work. End-to-end learning of codes have been studied in several settings, such as OFDM [60], one-bit quantization channels [61], and optical communications [62]. Taking a step further, exploiting the advantage of end-to-end trainability, several works have proposed learning frameworks for joint source channel coding [63, 64, 65].

VI. Discussion

Adapting recent advances in deep learning for building reliable communication systems is challenging but promising. From a deep learning perspective, the challenges are unique: we have access to potentially infinite training samples and the models are mathematically completely defined, but codebooks are exponentially large and we desire generalization in block lengths for efficient training. These challenges need to be handled via careful design of the neural network architecture, choice of training block length and SNR, and choice of the loss function and appropriate regularizers; we have seen the important role of insights from information and coding theory in successfully conducting these steps.

Looking Forward. In this article we have used vanilla RNNs to model both encoders and decoders of sequential codes. In deep learning literature, there is an important variant of vanilla RNNs – known as GRU [8] (and LSTM [9]) – which are proposed to mainly overcome the short memory limitations of RNNs. Specifically: GRU follows the same basic structure of an RNN as in Figure 6, where a hidden state h_k is updated over time based on h_{k-1} and In_k. The key difference is that GRU includes "gating": a reset gate r_k and an update gate z_k are used in updating the h_k. The hidden state h_k is now updated as:

$$\tilde{h}_k = \phi_h(W_h \text{In}_k + U_h(r_k \odot h_{k-1}) + b_h),$$
$$h_k = z_k \odot h_{k-1} + (1 - z_k) \odot \tilde{h}_k,$$

for trainable parameters W_h, U_h, b_h and an activation function ϕ_h, where $\odot$ refers to Hadamard product. A reset gate r_k

is defined as $r_k = \sigma_g(W_r \text{In}_k + U_r h_{k-1} + b_r)$ for trainable parameters W_r, U_r, b_r and an activation function σ_g. Similarly, an update gate z_k is parameterized as $z_k = \sigma_g(W_z \text{In}_k + U_z h_{k-1} + b_z)$ for trainable parameters W_z, U_z, b_z and an activation function σ_g. The role of the reset gate is to control the amount of influence of the previous hidden state in generating the candidate hidden state $\tilde{h}_k$. The role of the forget gate is to determine to which extent h_k just maintains h_{k-1} and reflect the new candidate hidden state $\tilde{h}_k$. What is important is these gates depend on the input and the previous state and thus the gating operation has the ability to change over time. In the context of sequential encoders, this means a GRU (and an LSTM) can represent a convolutional code with dynamically varying memory, and this variation can be made to functionally depend on an aspect of the communication scheme (example: feedback symbols). We can make an analogy to switched linear systems, where it is known that state dependent switching allows long term dependence of the state and output on the original input; this connection along with a learning theoretic study of GRUs and gated recurrent networks is made in [66, 67]. How to harness this added capability of GRUs in communication schemes (to further enhance the performance of sequential encoding and decoding schemes) is a promising open direction.

Another promising open direction involves studying deep learning architectures that could enhance and strengthen the other coding theoretic structures (cf. taxonomy in Figure 1); early work exploring the role of neural networks in decoding polar codes [68, 69, 70] is an example of this direction of research.

VII. Acknowledgement

This work is supported by NSF grants CNS-2002932 and CNS-2002664 and a gift from Intel.

References

[1] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[3] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[4] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[5] Lalit Bahl, John Cocke, Frederick Jelinek, and Josef Raviv. Optimal decoding of linear codes for minimizing symbol error rate (corresp.). *IEEE Transactions on information theory*, 20(2):284–287, 1974.

[6] Claude Berrou, Alain Glavieux, and Punya Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. 1. In *Proceedings of ICC'93-IEEE International Conference on Communications*, volume 2, pages 1064–1070. IEEE, 1993.

[7] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[8] Kyunghyun Cho, B van Merrienboer, Caglar Gulcehre, F Bougares, H Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[10] Xiao-An Wang and S. B. Wicker. An artificial neural net viterbi decoder. *IEEE Transactions on Communications*, 44(2):165–171, Feb 1996.

[11] Murat Hüsnü Sazlı and Can Işık. Neural network implementation of the bcjr algorithm. *Digital Signal Processing*, 17(1):353 – 359, 2007.

[12] Joan Bruna and X Li. Community detection with graph neural networks. *Stat*, 1050:27, 2017.

[13] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[14] Alex Nowak, Soledad Villar, Afonso S Bandeira, and Joan Bruna. Revised note on learning algorithms for quadratic assignment with graph neural networks. *arXiv preprint arXiv:1706.07450*, 2017.

[15] Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. Graph2seq: Scalable learning dynamics for graphs. *arXiv preprint arXiv:1802.04948*, 2018.

[16] Hyeji Kim, Yihan Jiang, Ranvir Rana, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. Communication algorithms via deep learning. In *International Conference on Learning Representations*, 2018.

[17] Meryem Benammar and Pablo Piantanida. Optimal training channel statistics for neural-based decoders. In *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, pages 2157–2161. IEEE, 2018.

[18] Ishay Be'ery, Nir Raviv, Tomer Raviv, and Yair Be'ery. Active deep decoding of linear codes. *arXiv preprint arXiv:1906.02778*, 2019.

[19] Yihan Jiang, Sreeram Kannan, Hyeji Kim, Sewoong Oh, Himanshu Asnani, and Pramod Viswanath. Deepturbo: Deep turbo decoder. In *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2019.

[20] Daniel Tandler, Sebastian Dörner, Sebastian Cammerer, and Stephan ten Brink. On recurrent neural networks for sequence-based processing in communications. In

53rd Asilomar Conference on Signals, Systems, and Computers, pages 537–543, 2019.

[21] Nir Shlezinger, Nariman Farsad, Yonina C Eldar, and Andrea J Goldsmith. Viterbinet: A deep learning based viterbi algorithm for symbol detection. *IEEE Transactions on Wireless Communications*, 2020.

[22] Tobias Gruber, Sebastian Cammerer, Jakob Hoydis, and Stephan ten Brink. On deep learning-based channel decoding. In *Information Sciences and Systems (CISS), 2017 51st Annual Conference on*, pages 1–6. IEEE, 2017.

[23] Eliya Nachmani, Yair Be'ery, and David Burshtein. Learning to decode linear codes using deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 341–346. IEEE, 2016.

[24] Eliya Nachmani, Elad Marciano, Loren Lugosch, Warren J Gross, David Burshtein, and Yair Be'ery. Deep learning methods for improved decoding of linear codes. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):119–131, 2018.

[25] Tomer Raviv, Nir Raviv, and Yair Be'ery. Data-driven ensembles for deep and hard-decision hybrid decoding. *arXiv preprint arXiv:2001.06247*, 2020.

[26] Mengke Lian, Fabrizio Carpi, Christian Häger, and Henry D Pfister. Learned belief-propagation decoding with simple scaling and snr adaptation. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 161–165. IEEE, 2019.

[27] Eliya Nachmani, Yaron Bachar, Elad Marciano, David Burshtein, and Yair Be'ery. Near maximum likelihood decoding with deep learning. *arXiv preprint arXiv:1801.02726*, 2018.

[28] Weihong Xu, Xiaohu You, Chuan Zhang, and Yair Be'ery. Polar decoding on sparse graphs with deep learning. In *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, pages 599–603. IEEE, 2018.

[29] Nghia Doan, Seyyed Ali Hashemi, Elie Ngomseu Mambou, Thibaud Tonnellier, and Warren J Gross. Neural belief propagation decoding of crc-polar concatenated codes. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[30] Fabrizio Carpi, Christian Häger, Marco Martalò, Riccardo Raheli, and Henry D Pfister. Reinforcement learning for channel coding: Learned bit-flipping decoding. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 922–929. IEEE, 2019.

[31] Amir Bennatan, Yoni Choukroun, and Pavel Kisilev. Deep learning for decoding of linear codes-a syndrome-based approach. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1595–1599. IEEE, 2018.

[32] Yihan Jiang, Hyeji Kim, Himanshu Asnani, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. Turbo autoencoder: Deep learning based channel codes for point-to-point communication channels. In *Advances in Neural Information Processing Systems*, pages 2754–2764, 2019.

[33] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[34] Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. Pacgan: The power of two samples in generative adversarial networks. In *Advances in neural information processing systems*, pages 1498–1507, 2018.

[35] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017.

[36] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[37] Erdal Arikan. Channel polarization: A method for constructing capacity-achieving codes. In *2008 IEEE International Symposium on Information Theory*, pages 1173–1177. IEEE, 2008.

[38] David JC MacKay and Radford M Neal. Near shannon limit performance of low density parity check codes. *Electronics letters*, 33(6):457–458, 1997.

[39] Simon S Du and Jason D Lee. On the power of over-parametrization in neural networks with quadratic activation. *arXiv preprint arXiv:1803.01206*, 2018.

[40] Shiyu Liang, Ruoyu Sun, and R Srikant. Revisiting landscape analysis in deep neural networks: Eliminating decreasing paths to infinity. *arXiv preprint arXiv:1912.13472*, 2019.

[41] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. Towards understanding the role of over-parametrization in generalization of neural networks. *arXiv preprint arXiv:1805.12076*, 2018.

[42] Timothy O'Shea and Jakob Hoydis. An introduction to deep learning for the physical layer. *IEEE Transactions on Cognitive Communications and Networking*, 3(4):563–575, 2017.

[43] Fayçal Ait Aoudia and Jakob Hoydis. End-to-end learning of communications systems without a channel model. In *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, pages 298–303. IEEE, 2018.

[44] Yihan Jiang, Hyeji Kim, Himanshu Asnani, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. Learn codes: Inventing low-latency codes via recurrent neural networks. *arXiv preprint arXiv:1811.12707*, 2018.

[45] Martin Klaus Muller, Fjolla Ademaj, Thomas Dittrich, Agnes Fastenbauer, Blanca Ramos Elbal, Armand Nabavi, Lukas Nagel, Stefan Schwarz, and Markus Rupp. Flexible multi-node simulation of cellular mobile communications: the Vienna 5G System Level Simulator. *EURASIP Journal on Wireless Communications and Networking*, 2018(1):17, September 2018.

[46] Bashar Tahir, Stefan Schwarz, and Markus Rupp. Ber comparison between convolutional, turbo, ldpc, and polar codes. In *2017 24th International Conference on Telecommunications (ICT)*, pages 1–7. IEEE, 2017.

[47] Moustafa Ebada, Sebastian Cammerer, Ahmed Elkelesh, and Stephan ten Brink. Deep learning-based polar code design. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 177–183. IEEE, 2019.

[48] Lingchen Huang, Huazi Zhang, Rong Li, Yiqun Ge, and Jun Wang. AI coding: Learning to construct error correction codes. *arXiv preprint arXiv:1901.05719*, 2019.

[49] A. Elkelesh, M. Ebada, S. Cammerer, and S. t. Brink. Decoder-tailored polar code design using the genetic algorithm. *IEEE Transactions on Communications*, 67(7):4521–4534, July 2019.

[50] Y. H. Kim, A. Lapidoth, and T. Weissman. Error exponents for the gaussian channel with noisy active feedback. In *2010 IEEE Information Theory Workshop on Information Theory (ITW 2010, Cairo)*, pages 1–3, Jan 2010.

[51] Nariman Farsad and Andrea Goldsmith. Neural network detection of data sequences in communication systems. *IEEE Transactions on Signal Processing*, 66(21):5663–5678, January 2018.

[52] Nariman Farsad, Nir Shlezinger, Andrea J. Goldsmith, and Yonina C. Eldar. Data-driven symbol detection via model-based machine learning. *arXiv preprint arXiv:2002.07806*, 2020.

[53] Hyeji Kim, Yihan Jiang, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. Deepcode: Feedback codes via deep learning. In *Advances in neural information processing systems*, pages 9436–9446, 2018.

[54] Claude Shannon. The zero error capacity of a noisy channel. *IRE Transactions on Information Theory*, 2(3):8–19, 1956.

[55] J Schalkwijk and Thomas Kailath. A coding scheme for additive noise channels with feedback–i: No bandwidth constraint. *IEEE Transactions on Information Theory*, 12(2):172–182, 1966.

[56] R. G. Gallager and B. Nakiboglu. Variations on a theme by schalkwijk and kailath. *IEEE Transactions on Information Theory*, 56(1):6–17, Jan 2010.

[57] HiSilicon Huawei. Performance evaluation of channel codes for control channel. *3GPP TSG-RAN WG1 #87 Reno, U.S.A., November 14-18, 2016*, R1-1611257.

[58] Yury Polyanskiy, H Vincent Poor, and Sergio Verdú. Channel coding rate in the finite blocklength regime. *IEEE Transactions on Information Theory*, 56(5):2307, 2010.

[59] Tomaso Erseghe. On the evaluation of the polyanskiy-poor-verdu converse bound for finite block-length coding in AWGN. 61:6578–6590, 01 2014.

[60] Alexander Felix, Sebastian Cammerer, Sebastian Dörner, Jakob Hoydis, and Stephan ten Brink. Ofdm-autoencoder for end-to-end learning of communications systems. *arXiv preprint arXiv:1803.05815*, 2018.

[61] Eren Balevi and Jeffrey G Andrews. Autoencoder-based error correction coding for one-bit quantization. *arXiv preprint arXiv:1909.12120*, 2019.

[62] Boris Karanov, Mathieu Chagnon, Félix Thouin, Tobias A. Eriksson, Henning Bülow, Domaniç Lavery, Polina Bayvel, and Laurent Schmalen. End-to-end deep learning of optical fiber communications. *J. Lightwave Technol.*, 36(20):4843–4855, Oct 2018.

[63] Kristy Choi, Kedar Tatwawadi, Tsachy Weissman, and Stefano Ermon. Necst: Neural joint source-channel coding. *arXiv preprint arXiv:1811.07557*, 2018.

[64] Nariman Farsad, Milind Rao, and Andrea Goldsmith. Deep learning for joint source-channel coding of text. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2326–2330. IEEE, 2018.

[65] E. Bourtsoulatze, D. B. Kurka, and D. Gündüz. Deep joint source-channel coding for wireless image transmission. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4774–4778, May 2019.

[66] Ashok Vardhan Makkuva, Sewoong Oh, Sreeram Kannan, and Pramod Viswanath. Learning in gated neural networks. *AISTATS*, 2020.

[67] Ashok Makkuva, Pramod Viswanath, Sreeram Kannan, and Sewoong Oh. Breaking the gridlock in mixture-of-experts: Consistent and efficient algorithms. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 4304–4313, 2019.

[68] Sebastian Cammerer, Fayçal Ait Aoudia, Sebastian Dörner, Maximilian Stark, Jakob Hoydis, and Stephan ten Brink. Trainable communication systems: Concepts and prototype. *arXiv preprint arXiv:1911.13055*, 2019.

[69] Sebastian Cammerer, Sebastian Dörner, Jakob Hoydis, and Stephan ten Brink. End-to-end learning for physical layer communications. In *The International Zurich Seminar on Information and Communication (IZS 2018) Proceedings*, pages 51–52. ETH Zurich, 2018.

[70] Sebastian Cammerer, Tobias Gruber, Jakob Hoydis, and Stephan Ten Brink. Scaling deep learning-based decoding of polar codes via partitioning. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE, 2017.