COMMUNICATION ALGORITHMS VIA DEEP LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Coding theory is a central discipline underpinning wireline and wireless modems that are the workhorses of the information age. Progress in coding theory is largely driven by individual human ingenuity with sporadic breakthroughs over the past century. In this paper we study whether it is possible to automate the discovery of coding and decoding algorithms via deep learning. We study a family of sequential codes parametrized by recurrent neural network (RNN) architectures. We show that creatively designed and trained RNN architectures can decode well known sequential codes such as the convolutional and turbo codes with close to optimal performance on the additive white Gaussian noise (AWGN) channel, which itself is achieved by breakthrough algorithms of our times (Viterbi and BCJR decoders, representing dynamic programing and forward-backward algorithms). We show strong generalizations, i.e., we train at a specific signal to noise ratio and block length but test at a wide range of these quantities, as well as robustness and adaptivity to deviations from the AWGN setting. Finally, we use the RNN architectures to design *new nonlinear* codes that represent a major progress in the long standing open problem of communicating reliably over the AWGN channel with noisy output feedback.

1 INTRODUCTION

Reliable digital communication, both wireline (ethernet, cable and DSL modems) and wireless (cellular, satellite, deep space), is a primary workhorse of the modern information age. A critical aspect of reliable communication involves the design of codes that allow transmissions to be robustly and (computationally efficiently) decoded under noisy conditions. This is the discipline of coding theory; over the past century and especially the past 70 years (since the birth of information theory (Shannon, 1948)) much progress has been made in the design of near optimal codes. Landmark codes include convolutional codes, turbo codes, low density parity check (LDPC) codes and, recently, 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 (Richardson & Urbanke, 2008).

The canonical setting is one of point-to-point reliable communication over the additive white Gaussian noise (AWGN) channel and performance of a code in this setting is its gold standard. The AWGN channel 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); again this is text book material (Tse & Viswanath, 2005). There are two long term goals in coding theory: (a) design of new, computationally efficient, codes that improve the state of the art (probability of correct reception) over the AWGN setting. Since the current codes already operate close to the information theoretic "Shannon limit", the emphasis is on *robustness* and *adaptability* to deviations from the AWGN settings (a list of channel models motivated by practical settings, (such as urban, pedestrian, vehicular) in the recent 5th generation cellular standard is available in Annex B of 3GPP TS 36.101. (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 (ranging from language (Mikolov et al., 2013) to vision (Russakovsky et al., 2015) to playing Go (Silver et al., 2016)). Motivated by these successes, we posit that deep learning methods can play a crucial role in solving both the aforementioned goals of coding theory and show that we can make significant progress on both these goals in this paper.

While the learning framework is clear and there is virtually unlimited training data available, there are two main challenges: (a) The space of codes is very vast and the sizes astronomical; for instance a rate 1/2 code over 100 information bits involves designing 2^{100} codewords in a 200 dimensional space. Computationally efficient encoding and decoding procedures are a must, apart from high reliability over the AWGN channel. (b) Generalization is highly desirable across block lengths and data rate that each work very well over a wide range of channel signal to noise ratios (SNR). In other words, one is looking to design a family of codes (parametrized by data rate and number of information bits) and their performance is evaluated over a range of channel SNRs.

In part due to these challenges, recent deep learing works on coding theory focus on decoding known codes using data-driven neural decoders and have limited themselves to small block lengths (O'Shea & Hoydis, 2017; Dörner et al., 2017; Gruber et al., 2017). Even here, generalization is difficult (for instance, even when nearly 90% of the codewords of a (8,16) polar code are exposed to the neural decoder (Gruber et al., 2017)). The main challenge is to restrict oneself to a class of codes that neural networks can naturally encode and decode. In this paper, we restrict ourselves to a class of *sequential* encoding and decoding schemes, of which convolutional and turbo codes are part of. These sequential coding schemes naturally meld with the family of recurrent neural network (RNN) architectures, which have recently seen large success in a wide variety of time-series tasks. The ancillary advantage of sequential schemes is that arbitrarily long information bits can be encoded and also at a large variety of coding rates.

Working within sequential codes parametrized by RNN architectures, we make the following contributions.

(1) Focusing on *convolutional codes* we aim to decode them on the AWGN channel using RNN architectures. Efficient optimal decoding of convolutional codes has represented historically fundamental progress in the broad arena of algorithms; optimal bit error decoding is achieved by the 'Viterbi decoder' (Viterbi, 1967) which is simply dynamic programming or Dijkstra's algorithm on a specific graph (the 'trellis') induced by the convolutional code. Optimal block error decoding is the BCJR decoder (Bahl et al., 1974) which is part of a family of forward-backward algorithms. While early work had shown that vanilla-RNNs are capable in *principle* of emulating both Viterbi and BCJR decoders (Wang & Wicker, 1996; Sazl & Ik, 2007) we show empirically, through a careful construction of RNN architectures and training methodology, that neural network decoding is possible at very near optimal performances (both bit error rate (BER) and block error rate (BLER)). The key point is that we train a RNN decoder at a *specific* SNR and over *short information bit* lengths (100 bits) and show *strong generalization* capabilities by testing over a wide range of SNR and block lengths (up to 10,000 bits). The specific training SNR is closely related to the Shannon limit of the AWGN channel at the rate of the code and provides strong information theoretic collateral to our empirical results.

(2) *Turbo codes* are naturally built on top of convolutional codes, both in terms of encoding and decoding. A natural generalization of our RNN convolutional decoders allow us to decode turbo codes at BER comparable to, and at certain regimes, even *better* than state of the art turbo decoders on the AWGN channel. That data driven, SGD-learnt, RNN architectures can decode comparably is fairly remarkable since turbo codes already operate near the Shannon limit of reliable communication over the AWGN channel.

(3) We show the afore-described neural network decoders for both convolutional and turbo codes are *robust* to variations to the AWGN channel model. We consider a problem of contemporary interest: communication over a "bursty" AWGN channel (where a small fraction of noise has much higher variance than usual) which models inter-cell interference in OFDM cellular systems (used in 4G and 5G cellular standards) or co-channel radar interference. We demonstrate empirically the neural network architectures can adapt to such variations and beat state of the art heuristics comfortably (despite evidence elsewhere that neural network are sensitive to models they are trained

on (Szegedy et al., 2013)). Via an innovative local perturbation analysis (akin to (Ribeiro et al., 2016)), we demonstrate the neural network to have learnt sophisticated preprocessing heuristics in engineering of real world systems (Li et al., 2013).

(4) So far we have restricted ourselves to neurally decoding fixed encoding schemes (i.e., convolutional and turbo codes). In the final section of this paper, we demonstrate new RNN-driven encoders (with matching decoders) that operate significantly better than state of the art on the AWGN channel with (noisy) output feedback. While feedback does not improve the Shannon capacity of the AWGN channel (Shannon, 1956), it is known to provide better reliability at finite block lengths (Schalkwijk & Kailath, 1966), although very sensitive to even tiny amounts of noise in the output feedback; more generally any linear code incorporating the noisy output feedback cannot achieve a non-zero reliable rate of communication (Kim et al., 2007b) – this is very troubling since all practical codes are linear and linear codes are known to achieve capacity (without feedback) (Elias, 1955). Our RNN parameterized encoders are inherently *nonlinear* and map information bits *directly* to real-valued transmissions. Their performance vastly improves the state of the art on the long standing open problem in information theory on communicating over the AWGN channel with noisy output feedback.

2 RNN DECODERS FOR SEQUENTIAL CODES

Among diverse families of coding scheme available in the literature, *sequential coding* schemes are particularly attractive. They (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; and (c) have a natural recurrent structure that is aligned with an established family of deep models, namely recurrent neural networks. We consider the basic sequential code known as *convolutional codes*, and provide a neural decoder that can be trained to achieve the optimal classification accuracy.

A standard example of a convolutional code is the *rate-1/2 Recursive Systematic Convolutional (RSC) code*. The encoder performs a forward pass on a recurrent network shown in Figure 1 on binary input sequence $\mathbf{b} = (b_1, \ldots, b_K)$, which we call *message bits*, with binary vector states $(s_1, \ldots, s_K)$ and binary vector outputs $(c_1, \ldots, c_K)$, which we call *transmitted bits* or a *codeword*. At time k with binary input $b_k \in \{0, 1\}$ and the state of a two-dimensional binary vector $s_k = (s_{k1}, s_{k2})$, the output is a two-dimensional binary vector $c_k = (c_{k1}, c_{k2}) = (2b_k - 1, 2(b_k \oplus s_{k1}) - 1) \in \{-1, 1\}^2$, 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., $s_1 = (0, 0)$.

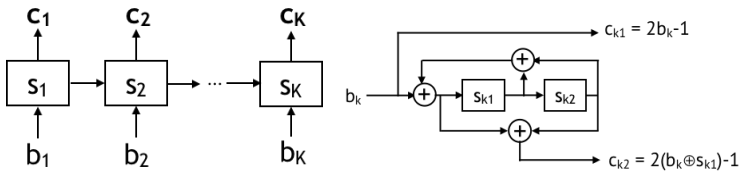


Figure 1: (Left) Sequential encoder is a recurrent network, (Right) One cell for a rate 1/2 RSC code

The $2K$ output bits are sent over a noisy channel, with the canonical one being the AWGN channel: the received binary vectors $\mathbf{y} = (y_1, \ldots, y_K)$, which are called the *received bits*, are $y_{ki} = c_{ki} + z_{ki}$ for all $k \in [K]$ and $i \in \{1, 2\}$, where z_{ki}'s are i.i.d. Gaussian with zero mean and variance σ^2. Decoding a received signal y refers to (attempting to) finding the maximum a posteriori (MAP) estimate. Due to the simple recurrent structure, efficient iterative schemes are available for finding the MAP estimate for convolutional codes (vit; Bahl et al., 1974). There are two MAP decoders depending on the error criterion in evaluating the performance: bit error rate (BER) or block error rate (BLER).

BLER counts the fraction of blocks that are wrongly decoded (assuming many such length-K blocks have been transmitted), and matching optimal MAP estimator is $\hat{\mathbf{b}} = \arg\max_{\mathbf{b}} \Pr(\mathbf{b}|\mathbf{y})$. Using dynamic programming, one can find the optimal MAP estimate in time linear in the block length K, which is called the *Viterbi algorithm*. BER counts the fraction of bits that are wrong, and

matching optimal MAP estimator is $\hat{b}_k = \arg\max_{b_k} \Pr(b_k|\mathbf{y})$, for all $k = 1, \cdots, K$. Again using dynamic programming, the optimal estimate can be computed in $O(K)$ time, which is called the *BCJR algorithm*.

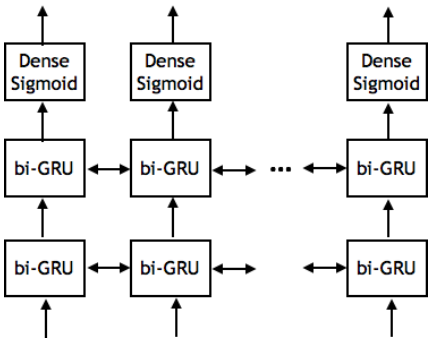
In both cases, the linear time optimal decoder crucially depends on the recurrent structure of the encoder. This structure can be represented as a hidden Markov chain (HMM), and both decoders are special cases of general efficient methods to solve inference problems on HMM using the principle of dynamic programming (e.g. belief propagation). These methods efficiently compute the exact posterior distributions in two passes through the network: the forward pass and the backward pass. Our first aim is to train a (recurrent) 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 can be matched by a neural network which only has access to the samples. The challenge lies in finding the right architecture and showing the right training examples.

Neural decoder for convolutional codes. We treat the decoding problem as a K-dimensional binary classification problem for each of the *message bits* b_k. The input to the decoder is a length-$2K$ sequence of received bits $\mathbf{y} \in \mathbb{R}^{2K}$ each associated with its length-K sequence of "true classes" $\mathbf{b} \in \{0,1\}^K$. The goal is to train a model to find an accurate sequence-to-sequence classifier. The input $\mathbf{y}$ is a noisy version of the class $\mathbf{b}$ according to the rate-1/2 RSC code defined in earlier in this section. We generate N training examples $(\mathbf{y}^{(i)}, \mathbf{b}^{(i)})$ for $i \in [N]$ according to this joint distribution to train our model.

We introduce a novel neural decoder for rate-1/2 RSC codes, we call N-RSC. It is two layers of bi-direction Gated Recurrent Units (bi-GRU) each followed by batch normalization units, and the output layer is a single fully connected sigmoid unit. Let W denote all the parameters in the model whose dimensions are shown in Figure 2, and $f_W(\mathbf{y}) \in [0,1]^K$ denote the output sequence. The k-th output $f_W(\mathbf{y})_k$ estimates the posterior probability $\Pr(b_k = 1|\mathbf{y})$, and we train the weights W to minimize the L_w error with respect to a choice of a loss function $\ell(\cdot,\cdot)$ specified below:

$$\mathcal{L} \quad = \quad \sum_{i=1}^{N} \sum_{k=1}^{K} \ell(f_W(\mathbf{y}^{(i)})_k, b_k^{(i)}) \ . \tag{1}$$

As the encoder is a recurrent network, it is critical that we use recurrent neural networks as a building block. Among several options of designing RNNs, we make three specific choices that are crucial in achieving the target accuracy: (i) bidirectional GRU as a building block instead of unidirectional GRU; (ii) 2-layer architecture instead of a single layer; and (iii) using batch normalization. As we show in Table 1 in Appendix C, unidirectional GRU fails because the underlying dynamic program requires bi-directional recursion of both forward pass and backward pass through the received sequence. A single layer bi-GRU fails to give the desired performance, and two layers is sufficient. We show how the accuracy depends on the number of layer in Table 1 in Appendix C. Batch normalization is also critical in achieving the target accuracy.



Layer	Output dimension
Input	(K, 2)
bi-GRU	(K, 400)
Batch Normalization	(K, 400)
bi-GRU	(K, 400)
Batch Normalization	(K, 400)
Dense (sigmoid)	(K, 1)

Figure 2: N-RSC: Neural decoder for RSC codes

Training. We propose two novel training techniques that improve accuracy of the trained model significantly. First, we propose a novel loss function guided by the efficient dynamic programming, that significantly reduces the number of training example we need to show. A natural L_2

loss (which gives better accuracy than cross-entropy in our problem) would be $\ell(f_W(\mathbf{y}^{(i)})_k, b_k^{(i)}) = (f_W(\mathbf{y}^{(i)})_k - b_k^{(i)})^2$. Recall that the neural network estimates the posterior $\Pr(b_k = 1|\mathbf{y}^{(i)})$, and the true label $b_k^{(i)}$ is a mere surrogate for the posterior, as typically the posterior distribution is simply not accessible. However, for decoding RSC codes, there exists efficient dynamic programming that can compute the posterior distribution exactly. This can significantly improve sample complexity of our training, as we are directly providing $\Pr(b_k = 1|\mathbf{y}^{(i)})$ as opposed to a sample from this distribution, which is $b_k^{(i)}$. We use a python implementation of BCJR in Taranalli (2015) to compute the posterior distribution exactly, and minimize the loss

$$\ell(f_W(\mathbf{y}^{(i)})_k, b_k^{(i)}) = (f_W(\mathbf{y}^{(i)})_k - \Pr(b_k = 1|\mathbf{y}^{(i)}))^2 \ . \tag{2}$$

Next, we provide a guideline for choosing the training examples that improve the accuracy. As it is natural to sample the training data and test data from the same distribution, one might use the same noise level for testing and training. However, this is not reliable as shown in Figure 3.

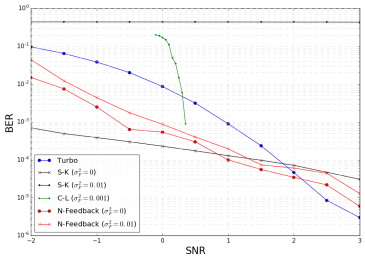


Figure 3: BER vs. test SNR for 0dB training and mismatched SNR training in decoding rate-1/2 RSC code of block length 100

Channel noise is measured by 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. For rate-1/2 RSC code, we propose using training data with noise level $\mathrm{SNR}_{\mathrm{train}} = \min\{\mathrm{SNR}_{\mathrm{test}}, 0\}$. Namely, matched training SNR if test SNR is below one, and otherwise fix training SNR at one independent of the test SNR. In Appendix E, we give a general formula for general rate-r codes, and provide an information theoretic justification and empirical evidences showing that this is near optimal choice of training data.

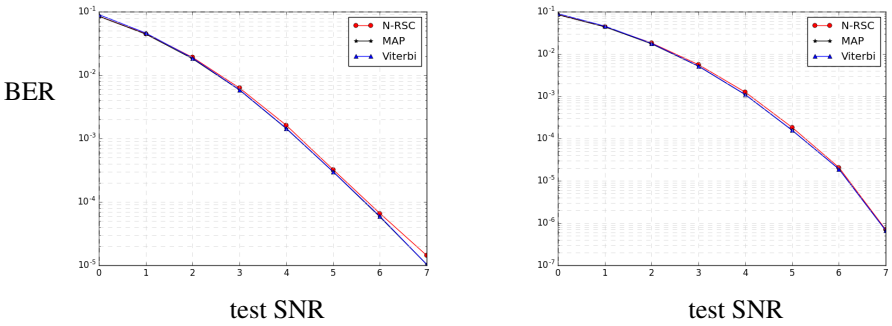


Figure 4: Rate-1/2 RSC code on AWGN. BER vs. SNR for block length (left) 100 and (right) 10,000

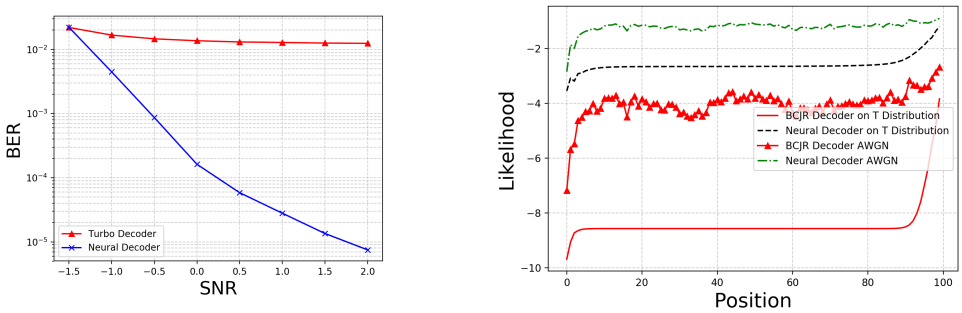
Performance. In Figure 4, for various test SNR, we train our N-RSC on randomly generated training data for rate-1/2 RSC code of block length 100 over AWGN channel with proposed training SNR of $\min\{\mathrm{SNR}_{\mathrm{test}}, 1\}$. We trained the decoder with Adam optimizer with learning rate 1e-3, batch size 200, and total number of examples is 12,000, and we use clip norm. On the left we show bit-error-rate when tested with length 100 RSC encoder, matching the training data. We show that N-RSC is able to learn to decode and achieve the optimal performance of the optimal dynamic programming (MAP decoder) almost everywhere. Perhaps surprisingly, we show on the right figure that we can use the neural decoder trained on length 100 codes, and apply it directly to codes of length $10,000$

and still meet the optimal performance. Note that we only give $12,000$ training examples, while the number of unique codewords is $2^{10,000}$. This shows that the proposed neural decoder (a) can generalize to unseen codeword; and (b) seamlessly generalizes to significantly longer block lengths. More experimental results including other types of convolutional codes are provided in Appendix A.

Turbo codes are naturally built out of convolutional codes (both encoder and decoder) and represent some of the most successful codes for the AWGN channel (Berrou et al., 1993). A corresponding stacking of multiple layers of the convolutional neural decoders leads to a natural neural turbo decoder which we show to match (and in some regimes *even beat*) the performance of standard state of the art turbo decoders on the AWGN channel; these details are available in Appendix B. Unlike the convolutional codes, the state of the art (message-passing) decoders for turbo codes are not the corresponding MAP decoders, so there is no contradiction in that our neural decoder would beat the message-passing ones. The training and architectural choices are similar in spirit to those of the convolutional code and are explored in detail in Appendix B.

3 NON-GAUSSIAN CHANNELS: ROBUSTNESS AND ADAPTIVITY

In the previous sections, we demonstrated that the neural decoder can perform as well as the turbo decoder. In practice, there are a wide variety of channel models that are suited for differing applications. Therefore, we test our neural decoder under some canonical channel models to see how robust and adaptive they are. Robustness refers to the ability of a decoder trained for a particular channel model to work well on a differing channel model *without re-training*. Adaptivity refers to the ability of the learning algorithm to adapt and *retrain* for differing channel models. In this section, we demonstrate that the neural turbo decoder is both adaptive and robust by testing on a set of non-Gaussian channel models.



(a) T-Distribution BER

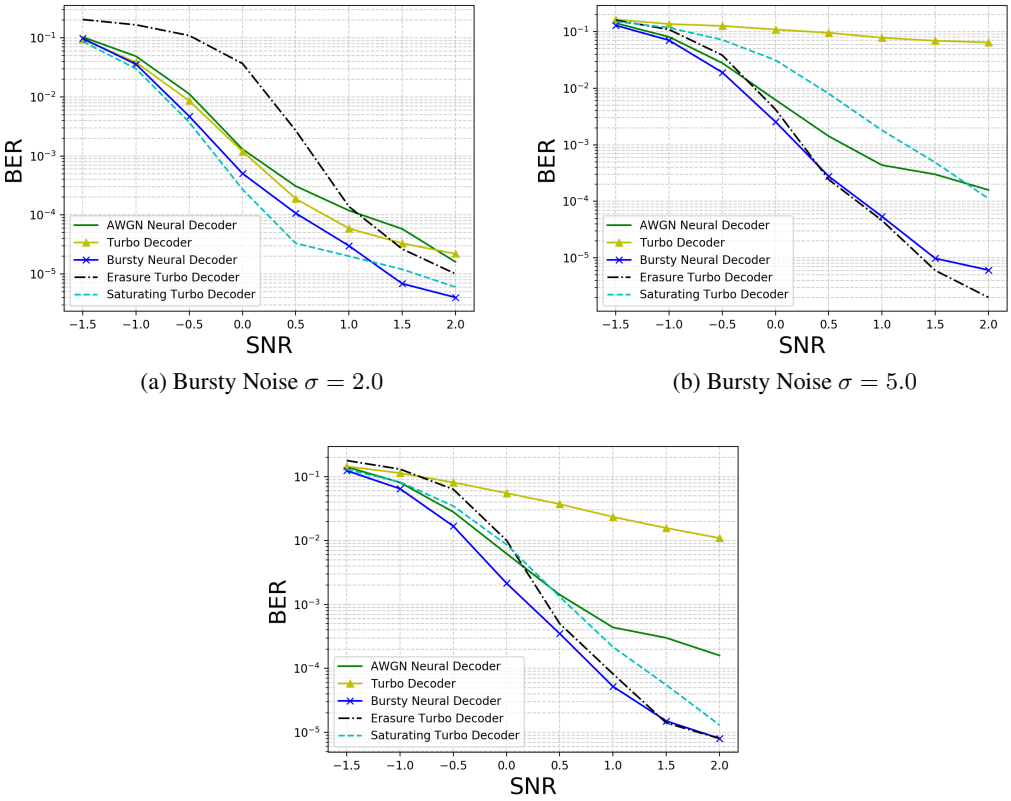
(b) Output Likelihood of BCJR and Neural Decoder

Figure 5: T-Distribution Performance

Robustness The robustness test is interesting from two directions, other than obvious practical value. Firstly, it is known from information theory that Gaussian noise is the worst case noise among all noise distributions with a given variance Cover & Thomas (2012); Lapidoth (1996). However, practical decoders can have much worse performance on channels different than the ones they are trained for. Second, this test poses an interesting challenge for neural decoders since deep neural networks are known to misclassify when tested against small adversarial perturbations Szegedy et al. (2013); Goodfellow et al. (2014). While we are not necessarily interested in adversarial perturbations to the input in this paper, it is important for the learning algorithm to be robust against differing noise distributions.

For the non-Gaussian channel, we choose the t-distribution family parameterized by parameter ν. We test the performance of both the neural and turbo decoder in this experiment when $\nu = 3$ in Figure 5a and observe that the neural decoder performs significantly better than the standard Turbo decoder (also see Figure 19a in Appendix F). In order to understand the reason for such a bad performance of the standard Turbo decoder, we plot the average output log-likelihood ratio (LLR) $\log p(b_k = 1) - \log p(b_k = 1)$ as a function of the bit position in Figure 5b, when the input is all-zero codeword. The main issue for the standard decoder is that the LLRs are not calculated

accurately (see Figure 19b in Appendix F): the LLR is exaggerated in the t-distribution while there is some exaggeration in the neural decoder as well, it is more modest in its prediction leading to more contained error propagation.



(a) Bursty Noise $\sigma = 2.0$

(b) Bursty Noise $\sigma = 5.0$

(c) Bursty Noise $\sigma = 3.5$

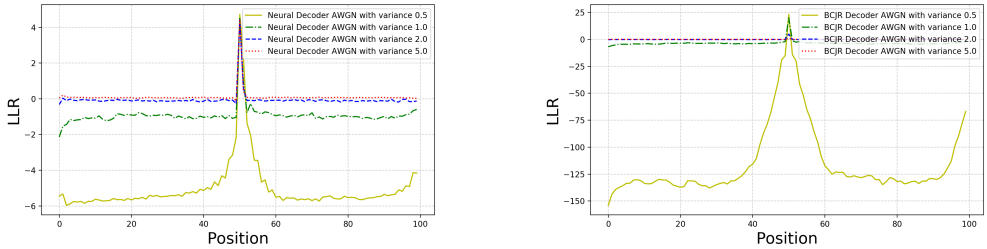
Figure 6: Neural Decoder Adaptivity under Different Bursty Noise Power

Adaptivity: A great advantage of neural channel decoder is that the neural network can learn a decoding algorithm even if the channel does not yield to a clean mathematical analysis. Consider a scenario where the transmitted signal is added with a Gaussian noise always, however, with a small probability, a further high variance noise is added. The channel model is mathematically described as follows, with y_i describing the received symbol and x_i denoting the transmitted symbol at time instant i: $y_i = x_i + z_i + w_i, \quad z_i \sim N(0, \sigma^2)$, and $w_i \sim N(0, \sigma_b^2)$ with probability ρ and $w_i = 0$ with probability $1 - \rho$, i.e., z_i denotes the Gaussian noise whereas w_i denotes the bursty noise.

This channel model accurately describes how radar signals (which are bursty) can create an interference for LTE in next generation wireless systems. This model has attracted attention in communications systems community due to its practical relevance Sanders et al. (2013); Sanders (2014). Under the aforesaid channel model, it turns out that standard Turbo coding decoder fails very badly Safavi-Naeini et al. (2015). The reason that the Turbo decoder cannot be modified in a straight-forward way is that the location of the bursty noise is a latent variable that needs to be jointly decoded along with the message bits. In order to combat this particular noise model, we fine-tune our neural decoder on this noise model, initialized from the AWGN neural decoder, and term it the bursty neural decoder. There are two state-of-the-art heuristics Safavi-Naeini et al. (2015): (a) erasure-thresholding: all LLR above a threshold are set to 0 (b) saturation-thresholding: all LLR above a threshold are set to the (signed) threshold.

We demonstrate the performance of our AWGN neural decoder (trained on Gaussian noise) as well as standard turbo decoder (for Gaussian noise) on this problem, shown in Figure 6 when $\sigma_b = 3.5, 2, 5$. We summarize the results of Figure 6: (1) standard turbo decoder not aware of bursty noise

will result in complete failure of decoding. (2) standard neural decoder still outperforms standard Turbo Decoder. (3) Bursty-neural-decoder outperforms Turbo Decoder using both state-of-the-art heuristics at $\sigma_b^2 = 3.5$ and obtains performance approaching that of the better of the two schemes at other variances.



(a) Neural Decoder Positional likelihood under Bursty Noise

(b) BCJR Decoder Positional likelihood under Bursty Noise

Figure 7: Positional BER and Likelihood under Bursty Noise added in the 50th position

Interpreting the Neural Decoder We try to interpret the action of the neural decoder trained under bursty noise. To do so, we look at the following simplified model, where $y_i = x_i + z_i + w_i$ where x_i, y_i, z_i are as before, but $w_i = B$ during the 50-th symbol in a 100-length codeword. We also fix the input codeword to be the all-zero codeword. We look at the average output LLR as a function of position for the one round of the neural decoder in Figure 7a and one round of BCJR algorithm in Figure 7b (the BER as a function of position is shown in Figure 20 in Appendix F). A negative LLR implies correct decoding at this level and a positive LLR implies incorrect decoding. It is evident that both RNN and BCJR algorithms make errors concentrated around the mid-point of the codeword. However, what is different between the two figures is that the scale of likelihoods of the two figures are quite different: the BCJR has a high sense of (misplaced) confidence, whereas the RNN is more modest in its assessment of its confidence. In the later stages of the decoding, the exaggerated sense of confidence of BCJR leads to an error propagation cascade eventually toggling other bits as well.

4 NEW CODES: AWGN CHANNEL WITH FEEDBACK

In previous sections, we have focused on decoding, via carefully designed and trained RNN architectures, of *existing* codes (convolutional and turbo) over the AWGN channel and variants. One reason is that for the AWGN channel it is known that linear codes are known to be optimal (Elias, 1955; Urbanke & Rimoldi, 1998). Designing new encoding schemes that outperform the strong state of the art (turbo, LDPC and polar codes) would represent major progress and remains an open problem. In this section, we consider an important communication scenario: AWGN channel with *output feedback*. The decoder receives $y_k = c_k + z_k$, $z_k \sim N(0, \sigma^2)$ at time k, which is fed back causally to the encoder; so the encoder has access to $\tilde{y}_k = y_k + w_k$, $w_k \sim N(0, \sigma_F^2)$ at time $k+1$, which it could use to design its transmissions c_t at times $t \geq k+1$. Here σ_F^2 is the variance of the noise in the feedback received by the encoder (modeling receiver quantization losses and feedback transmission losses).

Although output feedback (with or without noise) does not improve the Shannon capacity of the AWGN channel (Shannon, 1956), it is known to improve BER performance at finite block lengths. This was sharply brought to focus by the work of Schalkwijk & Kailath (1966) which considered the case of *perfect* output feedback. However, the celebrated scheme of Schalkwijk & Kailath (1966) is *very* sensitive to noise in the feedback (even precision errors when implemented in floating point arithmetic set the performance back dramatically). More generally, it is known that any linear scheme that incorporates the noisy output feedback with a finite rate cannot have BER be vanishingly small, no matter how long the blocklength is (Kim et al., 2007a; 2010). Practical efforts at incorporating output feedback has been less than successful, at times performing even worse than if the feedback were ignored. As such, designing codes for the AWGN channel that efficiently makes

use of the noisy output feedback is a long standing open problem in coding and communication theory; this is the focus of this section.

Architecture and Training. We focus on the setting of rate 1/3 and information block length of 50 for concreteness. The key innovation is in the design of a new encoding scheme that matches the benefits of noisy output feedback with the nonlinear representation capabilities of RNN architectures. The scheme progresses in two stages. In the first stage, the information bits are sent raw (uncoded) over the AWGN channel. In the second stage, the redundant bits are transmitted via a single-directional LSTM encoder which uses both the information bits and (delayed) output feedback from the earlier raw information bit transmissions. The sequential nature of the input to the LSTM encoder ensures we respect the causality involved in using the feedback. The decoder is a 2-layered bi-directional GRU. The architectures of encoder and decoder are shown in Figure 16,17 in Appendix D. Both the encoder and decoder are trained *jointly* using binary cross-entropy as the loss function over 10^7 examples. In training, we set the forward SNR equal to the test SNR and feedback SNR to be infinity, i.e.,noiseless feedback. We randomly initialize weights of the encoder and the decoder. We observed that random initialization of encoder-decoder was better than other initializations motivated by convolutional codes – codes that are designed for channels without feedback.

Results. The empirical BER findings are reported in Figure 8 in comparison to several baseline schemes: (a) the performance of a state of the art rate 1/3 turbo code that does not use the feedback information; (b) the performance of (Schalkwijk & Kailath, 1966) implemented using (default) 16 bit floating point arithmetic in Matlab and with noisy feedback; (c) recent codes for the feedback channel derived from (Chance & Love, 2011). The main finding is that our RNN driven encoder and decoder scheme significantly outperforms all the baselines. Significantly, we outperform even (Schalkwijk & Kailath, 1966) with *noisless* output feedback, which has been the gold standard in the literature for BER performance on the AWGN channel with output feedback.

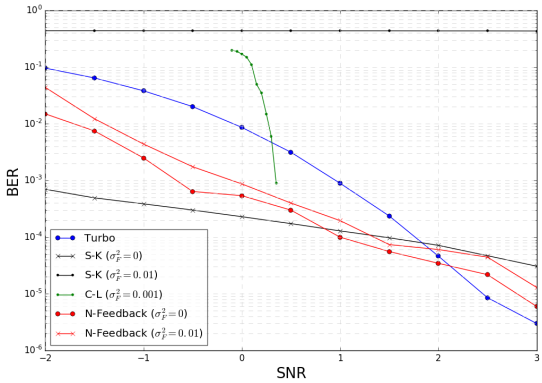


Figure 8: BER of N-Feedback and other coding schemes

5 Conclusion

In this paper we have demonstrated that appropriately designed and trained RNN architectures can 'learn' the landmark algorithms of Viterbi and BCJR decoding based on the strong generalization capabilities we demonstrate. This is similar in spirit to recent works on 'program learning' in the literature (Reed & De Freitas, 2015; Cai et al., 2017). In those works, the learning is assisted significantly by a low level program trace on an input; here we learn the Viterbi and BCJR algorithms only by end-to-end (input, output) training samples; we conjecture that this could be related to the strong "algebraic" nature of the Viterbi and BCJR algorithms. We have also demonstrated that appropriate RNN architectures can find new *encoding* schemes, as seen via the progress made on the challenging problem of communicating over the AWGN channel with noisy output feedback. These encoding and decoding capabilities of the RNN architectures suggest that new codes could be found on the AWGN channel itself and improve the state of the art (constituted by turbo, LDPC and polar codes). Also interesting is a new look at classical multi-terminal communication problems, including the relay and interference channels. Both are active areas of present research.

REFERENCES

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.

Claude Berrou, Alain Glavieux, and Punya Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. 1. In *Communications, 1993. ICC'93 Geneva. Technical Program, Conference Record, IEEE International Conference on*, volume 2, pp. 1064–1070. IEEE, 1993.

Jonathon Cai, Richard Shin, and Dawn Song. Making neural programming architectures generalize via recursion. *arXiv preprint arXiv:1704.06611*, 2017.

Z. Chance and D. J. Love. Concatenated coding for the awgn channel with noisy feedback. *IEEE Transactions on Information Theory*, 57(10):6633–6649, Oct 2011. ISSN 0018-9448. doi: 10.1109/TIT.2011.2165796.

Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

Sebastian Dörner, Sebastian Cammerer, Jakob Hoydis, and Stephan ten Brink. Deep learning-based communication over the air. *arXiv preprint arXiv:1707.03384*, 2017.

Peter Elias. Coding for noisy channels. In *IRE Convention record*, volume 4, pp. 37–46, 1955.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

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*, pp. 1–6. IEEE, 2017.

Y. H. Kim, A. Lapidoth, and T. Weissman. The gaussian channel with noisy feedback. In *2007 IEEE International Symposium on Information Theory*, pp. 1416–1420, June 2007a.

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)*, pp. 1–3, Jan 2010. doi: 10.1109/ITWKSPS.2010.5503226.

Young-Han Kim, Amos Lapidoth, and Tsachy Weissman. The gaussian channel with noisy feedback. In *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*, pp. 1416–1420. IEEE, 2007b.

Amos Lapidoth. Nearest neighbor decoding for additive non-gaussian noise channels. *IEEE Transactions on Information Theory*, 42(5):1520–1529, 1996.

Junyi Li, Xinzhou Wu, and Rajiv Laroia. *OFDMA mobile broadband communications: A systems approach*. Cambridge University Press, 2013.

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*, pp. 3111–3119, 2013.

Timothy J. O'Shea and Jakob Hoydis. An introduction to machine learning communications systems. *CoRR*, abs/1702.00832, 2017. URL `http://arxiv.org/abs/1702.00832`.

Scott Reed and Nando De Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 1135–1144, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939778. URL `http://doi.acm.org/10.1145/2939672.2939778`.

Tom Richardson and Ruediger Urbanke. *Modern coding theory*. Cambridge university press, 2008.

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.

Hossein-Ali Safavi-Naeini, Chittabrata Ghosh, Eugene Visotsky, Rapeepat Ratasuk, and Sumit Roy. Impact and mitigation of narrow-band radar interference in down-link lte. In *Communications (ICC), 2015 IEEE International Conference on*, pp. 2644–2649. IEEE, 2015.

Frank H Sanders, John E Carroll, Geoffrey A Sanders, and Robert L Sole. Effects of radar interference on lte base station receiver performance. *NTIA, US Dept. of Commerce*, 2013.

Geoffrey A Sanders. *Effects of radar interference on LTE (FDD) eNodeB and UE receiver performance in the 3.5 GHz band*. US Department of Commerce, National Telecommunications and Information Administration, 2014.

Murat Hsn Sazl and Can Ik. Neural network implementation of the bcjr algorithm. *Digital Signal Processing*, 17(1):353 – 359, 2007. ISSN 1051-2004. doi: https://doi.org/10.1016/j.dsp.2005.12.002. URL http://www.sciencedirect.com/science/article/pii/S1051200406000029.

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.

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

Claude E Shannon. A mathematical theory of communication, part i, part ii. *Bell Syst. Tech. J.*, 27: 623–656, 1948.

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.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Veeresh Taranalli. Commpy: Digital communication with python, version 0.3.0. available at https://github.com/veeresht/commpy. 2015.

David Tse and Pramod Viswanath. *Fundamentals of wireless communication*. Cambridge university press, 2005.

Rüdiger Urbanke and Bixio Rimoldi. Lattice codes can achieve capacity on the awgn channel. *IEEE transactions on Information Theory*, 44(1):273–278, 1998.

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

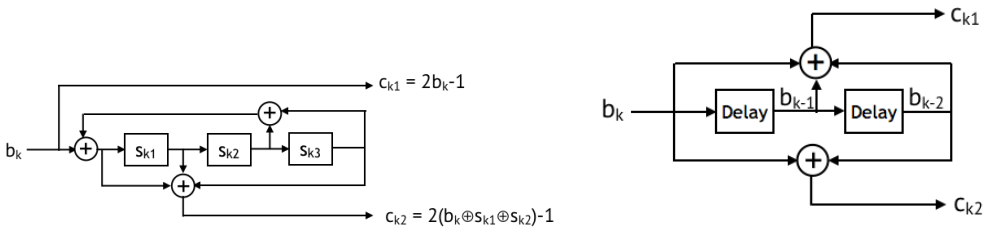
Xiao-An Wang and S. B. Wicker. An artificial neural net viterbi decoder. *IEEE Transactions on Communications*, 44(2):165–171, Feb 1996. ISSN 0090-6778.

APPENDIX

A NEURAL DECODER FOR OTHER CONVOLUTIONAL CODES

The rate-1/2 RSC code introduced in Section 2 is one example of many convolutional codes. In this section, we show empirically that neural decoders can be trained to decode other types of convolutional codes as well as MAP decoder. We consider the following two convolutional codes.

Unlike the rate-1/2 RSC code in Section 2, the convolutional code in Figure 9(a) is not recursive, i.e., state does not have a feedback. Also, it is non-systematic, i.e., the message bits can not be seen



(a) Rate-1/2 RSC code with state dimension 3

(b) Rate-1/2 non-recursive non-systematic convolutional code

Figure 9: Examples of Rate 1/2 Convolutional code

immediately from the coded bits. The convolutional code in Figure 9(b) is another type of rate-1/2 RSC code with a larger state dimension (dimension 3 instead of 2).

Figure 9 show the architecture of neural network we used for the convolutional codes in Figure 9. For the code in Figure 9(a), we used the exact same architecture we used for the rate-1/2 RSC code in Section 2. For the code in Figure 9(b), we used a larger network (LSTM instead of GRU and 800 hidden units instead of 400). This is due to the increased state dimension in the encoder.

Layer	Output dimension
Input	(K, 2)
bi-GRU	(K, 400)
Batch Normalization	(K, 400)
bi-GRU	(K, 400)
Batch Normalization	(K, 400)
Dense (sigmoid)	(K, 1)

Layer	Output dimension
Input	(K, 2)
bi-LSTM	(K, 800)
Batch Normalization	(K, 800)
bi-LSTM	(K, 800)
Batch Normalization	(K, 800)
Dense (sigmoid)	(K, 1)

Figure 10: Neural decoders for convolutional codes in (left) Figure 9 (a) and (right) Figure 9 (b)

For training of neural decoder in Figure 9(a), we used 12000 training examples of block length 100 with fixed SNR $0dB$. For training convolutional code (b), we used 48000 training examples of block length 500. We set batch size 200 and clip norm. The convolutional code (b) has a larger state space.

Performance. In Figures 11 , we show the BER and BLER of the trained neural decoder for convolutional code in Figure 9(a) under various SNRs and block lengths. As we can see from these figures, neural decoder trained on one SNR (0dB) and short block length (100) can be generalized to decoding as good as MAP decoder under various SNRs and block lengths. Similarly in Figure 12, we show the BER and BLER performances of trained neural decoder for convolutional code in Figure 9(b), which again shows the generalization capability of the trained neural decoder.

B NEURAL DECODER FOR TURBO CODES

Turbo codes, also called parallel concatenated convolutional codes, are popular in practice as they significantly outperform RSC codes. We provide a neural decoder for turbo codes using multiple layers of neural decoder we introduced for RSC codes. An example of rate-1/3 turbo code is shown in Figure 13. Two identical rate-1/2 RSC encoders are used, encoder 1 with original sequence $\mathbf{b}$ as input and encoder 2 with a randomly permuted version of $\mathbf{b}$ as input. Interleaver performs the random permutation. As the first output sequence $c_1(1)$ of encoder 1 is identical to the output sequence $c_1(2)$ of encoder 2, and hence redundant. So the sequence $c_1(2)$ is thrown away, and the rest of the sequences $(c_1(1), c_2(1), c_2(2))$ are transmitted; hence, rate is 1/3.

The sequences $(c_1(1), c_2(1), c_2(2))$ are transmitted over AWGN channel, and the noisy received sequences are $(y_1(1), y_2(1), y_2(2))$. Due to the interleaved structure of the encoder, MAP decoding is computationally intractable. Instead, an iterative decoder known as *turbo decoder* is used in practice, which uses the RSC MAP decoder (BCJR algorithm) as a building block. At first iteration the standard BCJR estimates the posterior $\Pr(b_k | y_1(1), y_2(1))$ with uniform prior on b_k for all $k \in$

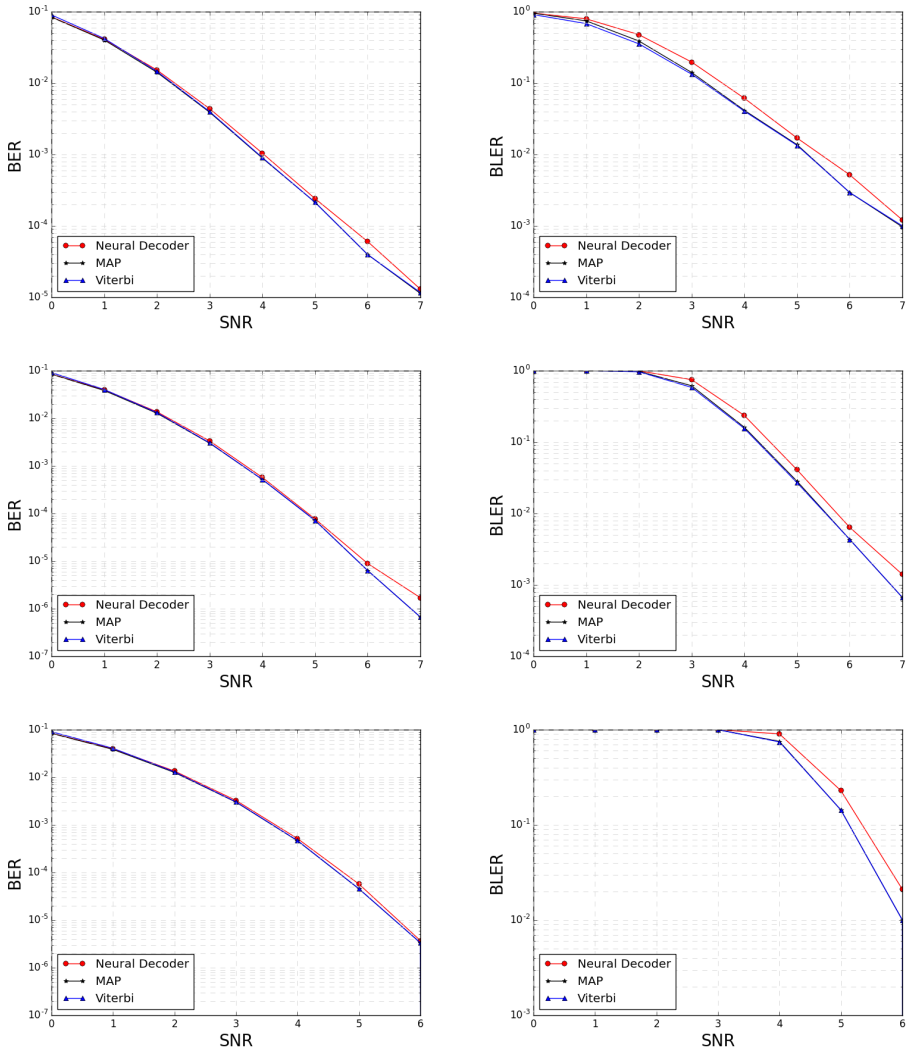


Figure 11: Rate-1/2 RSC code in Figure 9(a) on AWGN. (Left) BER and (Right) BLER vs. SNR for block length 100, 1000, and 10,000

$[K]$. Next, BCJR estimates $\Pr(b_k|\pi(y_1(1)), y_2(2))$ with the interleaved sequence $\pi(y_1(1))$, but now takes the output of the first layer as a prior on b_k's. This process is repeated, refining the belief on what the codewords b_k's are, until convergence and an estimation is made in the end for each bit.

Training. We propose a neural decoder for turbo codes that we call N-Turbo in Figure 13. Following the deep layered architecture of the turbo decoder, we stack layers of a variation of our N-RSC decoder, which we call N-BCJR. However, end-to-end training (using examples of the input sequence $\mathbf{y}^{(i)}$'s and the message sequence of $\mathbf{b}^{(i)}$'s) of such a deep layers of recurrent architecture is challenging. We propose first training each layer separately, use these trained models as initializations, and train the deep layered neural decoder of N-Turbo starting from these initialized weights.

We first explain our N-BCJR architecture, which is a new type of N-RSC that can take flexible bit-wise prior distribution as input. Previous N-RSC we proposed is customized for uniform prior distribution. The architecture is similar to the one for N-RSC. The main difference is input size (3 instead of 2) and the type of RNN (LSTM instead of GRU). To generate N training examples of $\{(noisycodeword, prior), posterior\}$, we generate $N/12$ examples of turbo codes. Then we ran turbo decoder for 12 component decoding - and collect input output pairs from the 12 intermediate steps of Turbo decoder, implemented in python Taranalli (2015) shown in Figure 14.

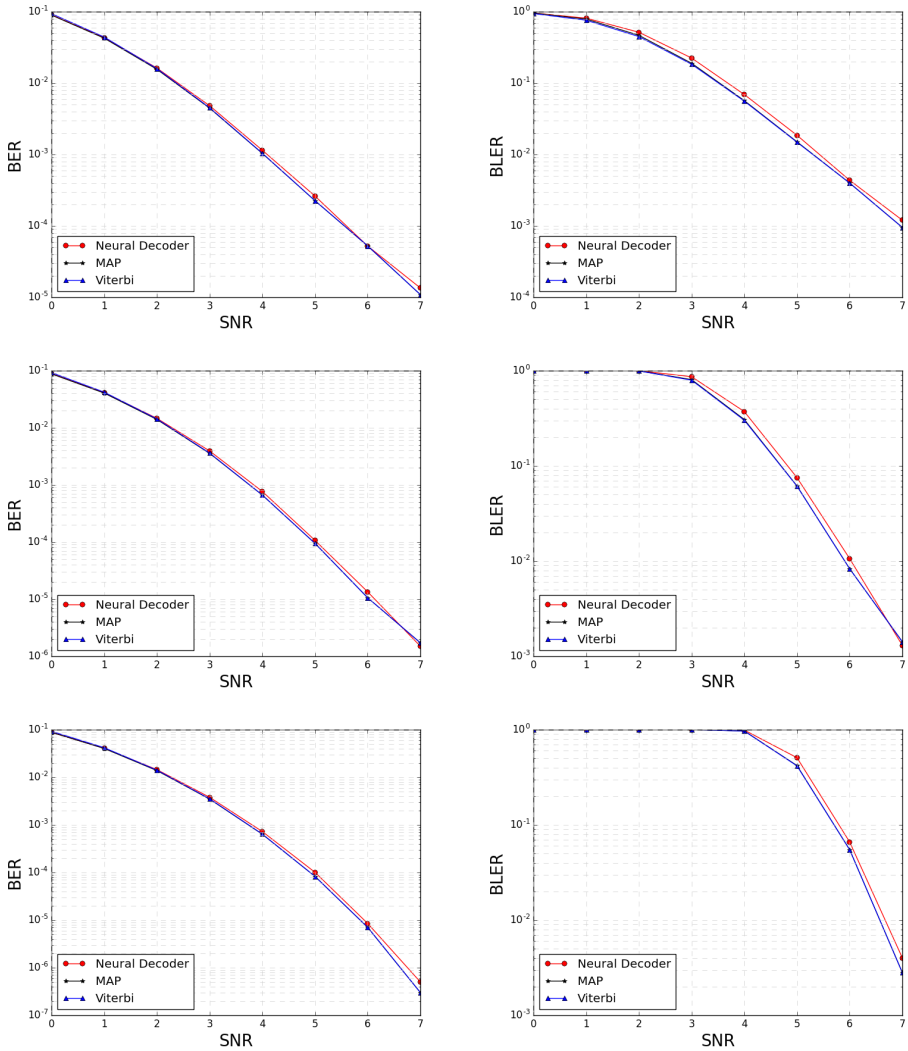
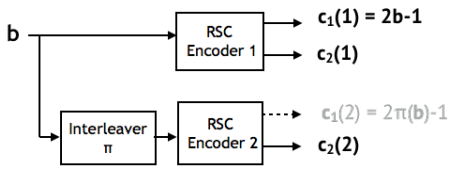


Figure 12: Rate-1/2 convolutional code in Figure 9(b) on AWGN. (Left) BER and (Right) BLER vs. SNR for block length 100, 1000, and 10,000

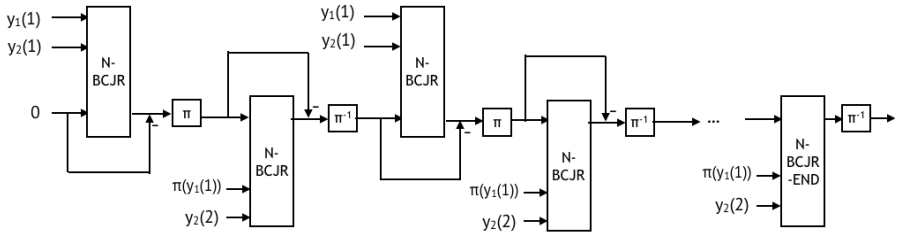
We train with codes with blocklength 100 at fixed SNR -1dB. We use mean squared error in (2) as a cost function.

To generate training examples with non-zero priors, i.e. example of a triplet (prior probabilities $\{\Pr(b_k)\}_{k=1}^{K}$, a received sequence $\mathbf{y}$, and posterior probabilities of the message bits $\{\Pr(b_k|\mathbf{y})\}_{k=1}^{K}$), we use intermediate layers of a turbo decoder. We run turbo decoder, and in each of the intermediate layers, we take as an example the triplet: the input prior probability, the input sequence, and the output of the BJCR layer. We fix training SNR to be -1dB. We stack 6 layers of BCJR decoder with interleavers in between. The last layer of our neural decoder is trained slightly differently to output the estimated message bit and not the posterior probability. Accordingly, we use binary crossentropy loss of as a cost function. We train each N-BCJR layer with 2,000 examples of length 100 turbo encoder, and in the end-to-end training of N-Turbo, we train with 1,000 examples of length 1,000 turbo encoder. We train with 10 epochs and ADAM optimizer with learning rate 0.001. For the end-to-end training, we again use a fixed SNR of noise (-1dB), and test on various SNRs. The choice of training SNR is discussed in detail in the Appendix E.

Performance. As can be seen in Figure 15, the proposed N-Turbo meets the performance of turbo decoder for block length 100, and in some cases, for test SNR$=$ 2, it achieves a higher accuracy.



(a) rate-1/3 turbo encoder



(b) Proposed neural turbo decoder (N-Turbo)

Figure 13: rate-1/3 turbo encoder (top) and neural turbo decoder N-Turbo (bottom)

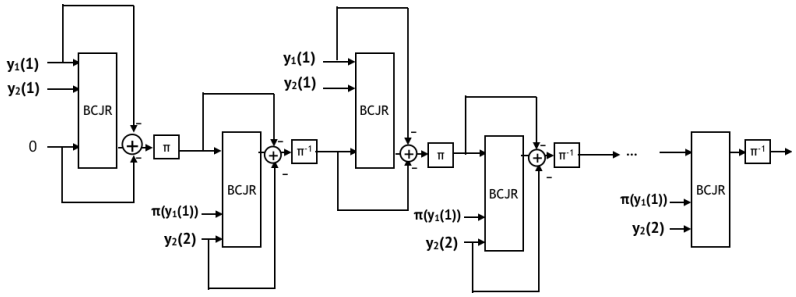


Figure 14: Turbo decoder

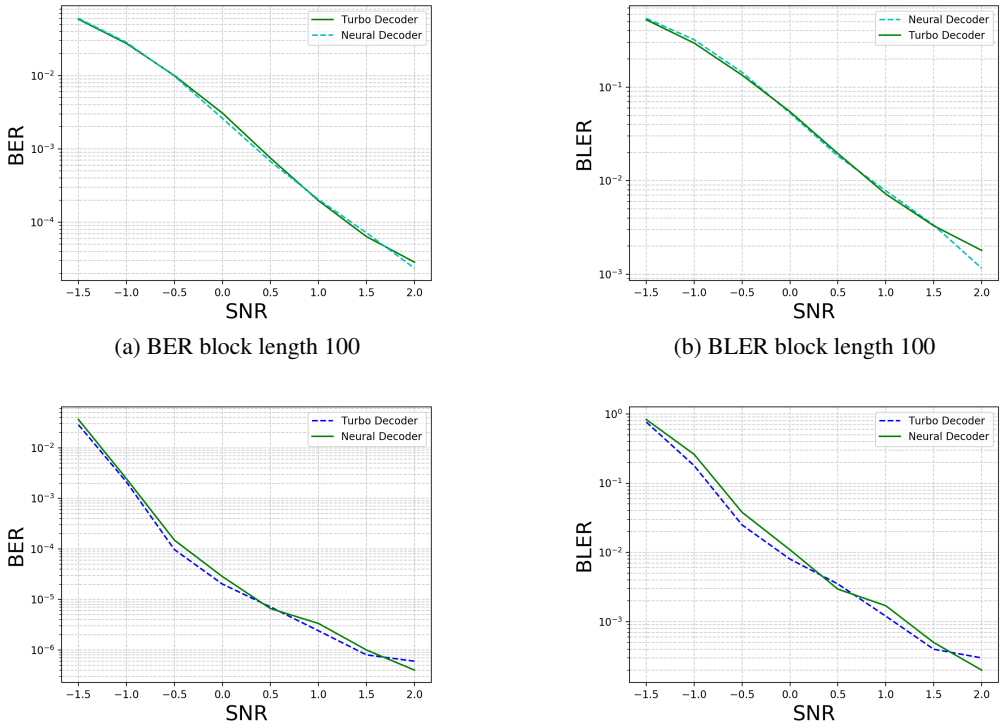
Similar to N-RSC, N-Turbo generalizes to unseen codewords, as we only show $3,000$ examples in total. It also seamlessly generalizes in the test SNR, as training SNR is fixed at -1dB.

C OTHER NEURAL NETWORK ARCHITECTURES FOR N-RSC AND N-BCJR

In this section, we show the performances of neural networks of various recurrent network architectures in decoding rate-1/2 RSC code and in learning BCJR algorithm with non-zero priors. Table 1 shows the BER of various types of recurrent neural networks trained under the same condition as in N-RSC (120000 example, code length 100). We can see that BERs of the 1-layered RNN and single-directional RNN are order-wise worse than the one of 2-layered GRU (N-RSC), and two layers is sufficient. Table 2 shows the performance of neural networks of various recurrent network architectures in BCJR training. Again, we can see that 2-layers are needed and single directional RNN does not work as well as bi-directional RNNs.

D ARCHITECTURE OF RATE-1/3 NEURAL ENCODER AND DECODER FOR NOISY FEEDBACK CHANNEL

In this section, we show the architectures for rate-1/2 recurrent neural encoder and decoder for channels with noisy feedback in Figure 16 and 17. The number of message bits is denoted by K. The encode maps K message bits to a codeword of length $3K$. Note that unlike in RSC or Turbo codes, the codewords are not restricted to be binary.



(a) BER block length 100

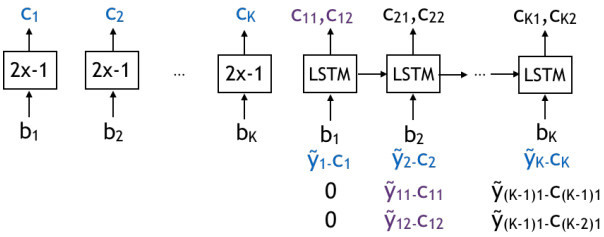
(b) BLER block length 100

(c) BER block length 1000

(d) BLER block length 1000

Figure 15: N-Turbo matches the performance of the Turbo decoder on AWGN.

The encoding process has two phases. In the first phase, the encoder simply transmits the raw message bits without any coding and collects the feedback $\tilde{y}_1, \cdots, \tilde{y}_K$. In the second phase, the encoder generates coded sequence of length $2K$ through a single directional LSTM. The input to the kth LSTM cell is of size 4: b_k, $\tilde{y}_k - c_k$ and the most recent two noise feedback $\tilde{y}_{(k-1)1} - c_{(k-1)1}$ and $\tilde{y}_{(k-1)2} - c_{(k-1)2}$ which are available fed back by the receiver (Notation: $\tilde{y}_{(ij)}$ denotes the noisy version of c_{ij}). This architecture naturally assures the causality of using feedback in the encoder, i.e., coded bit at time k depends only on the feedback $\tilde{y}_1, \cdots, \tilde{y}_{k-1}$. For the decoder, we use 2 layered bidirectional GRU.



Layer	Output dimension
Input	(K, 4)
LSTM	(K, 100)
Dense (sigmoid)	(K, 2)

Figure 16: Rate-1/3 recurrent neural encoder for noisy feedback channel

E GUIDELINES FOR CHOOSING THE TRAINING SNR FOR NEURAL DECODERS

As it is natural to sample the training data and test data from the same distribution, one might use the same noise level for testing and training. However, this matched SNR is not reliable as shown in Figure 3. We give an analysis that predicts the appropriate choice of training SNR that might be

Depth	BER (at 4dB)	N (Training examples)	Hidden units
bi-LSTM-1	0.01376	12e+5	200
bi-GRU-1	0.01400	12e+5	200
uni-GRU-2	0.01787	12e+5	200
bi-RNN-2	0.05814	12e+5	200
bi-GRU-2	0.00128	12e+5	200
bi-GRU-3	0.00127	12e+5	200
bi-GRU-4	0.00128	12e+5	200
bi-GRU-5	0.00132	12e+5	200

Table 1: BER (at 4dB) of trained neural decoders with different number/type of RNN layers on rate-1/2 RSC codes (blocklength 100) at SNR 4dB

BCJR-like RNN Performance			
Model	Number of Hidden Unit	BCJR Val MSE	Turbo BER (Turbo 6 iters: 0.002)
BD-1-LSTM	100	0.0031	0.1666
BD-1-GRU	100	0.0035	0.1847
BD-1-RNN	100	0.0027	0.1448
BD-1-LSTM	200	0.0031	0.1757
BD-1-GRU	200	0.0035	0.1693
BD-1-RNN	200	0.0024	0.1362
SD-1-LSTM	100	0.0033	0.1656
SD-1-GRU	100	0.0034	0.1827
SD-1-RNN	100	0.0033	0.2078
SD-1-LSTM	200	0.0032	0.137
SD-1-GRU	200	0.0033	0.1603
SD-1-RNN	200	0.0024	1462
BD-2-LSTM	100	4.4176e-04	0.1057
BD-2-GRU	100	1.9736e-04	0.0128
BD-2-RNN	100	7.5854e-04	0.0744
BD-2-LSTM	200	1.5917e-04	0.01307
BD-2-GRU	200	1.1532e-04	0.00609
BD-2-RNN	200	0.0010	0.11229
SD-2-LSTM	100	0.0023	0.1643
SD-2-GRU	100	0.0026	0.1732
SD-2-RNN	100	0.0023	0.1614
SD-2-LSTM	200	0.0023	0.1643
SD-2-GRU	200	0.0023	0.1582
SD-2-RNN	200	0.0023	0.1611

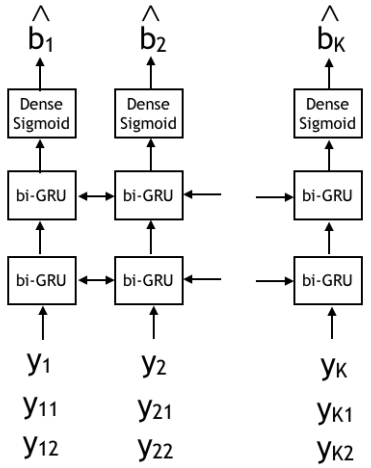
Table 2: MSE of trained neural models with different number/type of RNN layers in learning BCJR algorithm with non-zero priors

different from testing SNR, and justify our choice via comparisons over various pairs of training and testing SNRs.

We conjecture that the optimal training SNR that gives best BER for a target testing SNR depends on the coding rate. A coding rate is defined as the ratio between the length of the message bit sequence K and the length of the transmitted codeword sequence $\mathbf{c}$. The example we use in this paper is a rate $r = 1/2$ code with length of $\mathbf{c}$ equal to $2K$. For a rate r code, we propose using training SNR according to

$$\mathrm{SNR}_{\mathrm{train}} = \min\{\mathrm{SNR}_{\mathrm{test}}, 10\log_{10}(2^{2r} - 1)\} \ , \tag{3}$$

and call the knee of this curve $f(r) = 10\log_{10}(2^{2r} - 1)$ a threshold. In particular, this gives $\mathrm{SNR}_{\mathrm{train}} = \min\{\mathrm{SNR}_{\mathrm{test}}, 0\}$ for rate $1/2$ codes. In Figure 18 left, we train our neural decoder for RSC encoders of varying rates of $r \in \{1/2, 1/3, 1/4, 1/5, 1/6, 1/7\}$ whose corresponding $f(r) =$



Layer	Output dimension
Input	(K, 3)
bi-GRU	(K, 400)
Batch Normalization	(K, 400)
bi-GRU	(K, 400)
Batch Normalization	(K, 400)
Fully connected (sigmoid)	(K, 1)

Figure 17: Recurrent neural decoder for rate-1/3 encoder in Figure 16 under noisy feedback channel

$\{0, -2.31, -3.82, -4.95, -5.85, -6.59\}$. $f(r)$ is plotted as a function of the rate r in Figure 18 right panel. Compared to the grey shaded region of empirically observed region of training SNR that achieves the best performance, we see that it follows the theoretical prediction up to a small shift. The figure on the left shows empirically observed best SNR for training at each testing SNR for various rate r codes. We can observe that it follows the trend of the theoretical prediction of a curve with a knee. Before the threshold, it closely aligns with the 45-degree line $\mathrm{SNR}_{\mathrm{train}} = \mathrm{SNR}_{\mathrm{test}}$. around the threshold, the curves become constant functions.

We derive the formula in (3) in two parts. When the test SNR is below the threshold, then we are targeting for bit error rate (and similarly the block error rate) of around $10^{-1} \sim 10^{-2}$. This implies that significant portion of the testing examples lie near the decision boundary of this problem. Hence, it makes sense to show matching training examples, as significant portion of the training examples will also be at the boundary, which is what we want in order to maximize the use of the samples. On the other hand, when we are above the threshold, our target bit-error-rate can be significantly smaller, say 10^{-6}. In this case, most of the testing examples are *easy*, and only a very small proportion of the testing examples lie at the decision boundary. Hence, if we match training SNR, most of the examples will be wasted. Hence, we need to show those examples at the decision boundary, and we propose that the training examples from SNR $10\log_{10}(2^{2r}-1)$ should lie near the boundary. This is a crude estimate, but effective, and can be computed using the capacity achieving random codes for AWGN channels and the distances between the codes words at capacity. Capacity is a fundamental limit on what rate can be used at a given test SNR to achieve small error. In other words, for a given test SNR over AWGN channel, Gaussian capacity gives how closely we can pack the codewords (the classes in our classification problem) so that they are as densely packed as possible. This gives us a sense of how decision boundaries (as measured by the test SNR) depend on the rate. It is given by the Gaussian capacity $\mathrm{rate} = 1/2\log(1 + SNR)$. Translating this into our setting, we set the desired threshold that we seek.

F SUPPLEMENTARY FIGURES FOR SECTION 3

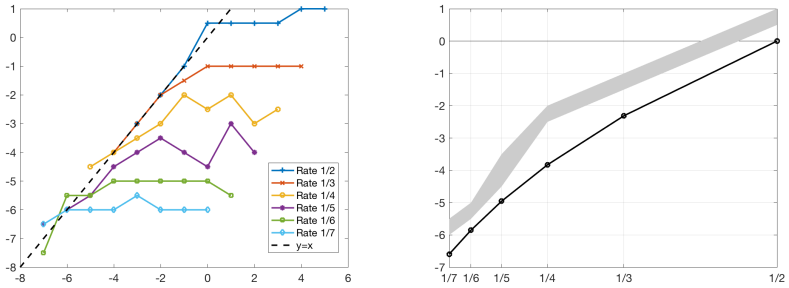
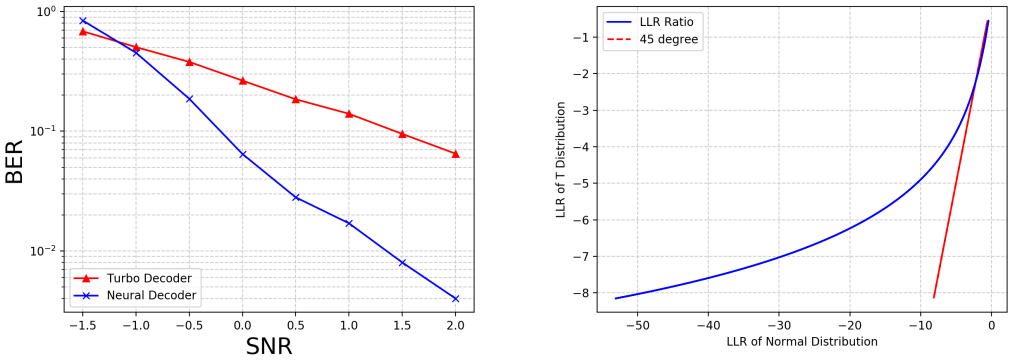


Figure 18: (Left) Best training SNR vs. Test SNR (Right) Best training SNR vs. code rate



(a) T-Distribution BLER

(b) Gaussian LLR verses T-distribution LLR

Figure 19: T-Distribution Performance

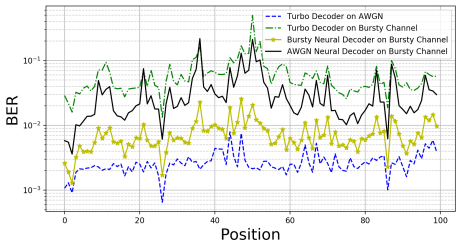


Figure 20: Turbo Decoder Positional BER log scale