DEEPTURBO : Deep Turbo Decoder

Yihan Jiang
ECE Department
University of Washington
Seattle, United States
yij021@uw.edu

Hyeji Kim
Samsung AI Center Cambridge
Cambridge, United Kingdom
hkim1505@gmail.com

Himanshu Asnani
ECE Department
University of Washington
Seattle, United States
asnani@uw.edu

Sreeram Kannan
ECE Department
University of Washington
Seattle, United States
ksreeram@ee.washington.edu

Sewoong Oh
Allen School of Computer
Science & Engineering
University of Washington
Seattle, United States
sewoong@cs.washington.edu

Pramod Viswanath
ECE Department
University of Illinois at Urbana
Champaign
Illinois, United States
pramodv@illinois.edu

Abstract—

Present-day communication systems routinely use codes that approach the channel capacity when coupled with a computationally efficient decoder. However, the decoder is typically designed for the Gaussian noise channel, and is known to be sub-optimal for non-Gaussian noise distribution. Deep learning methods offer a new approach for designing decoders that can be trained and tailored for arbitrary channel statistics. We focus on Turbo codes, and propose (DEEPTURBO), a novel deep learning based architecture for Turbo decoding.

The standard Turbo decoder (TURBO) iteratively applies the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm with an interleaver in the middle. A neural architecture for Turbo decoding, termed (NEURALBCJR), was proposed recently. There, the key idea is to create a module that *imitates* the BCJR algorithm using supervised learning, and to use the interleaver architecture along with this module, which is then fine-tuned using end-to-end training. However, knowledge of the BCJR algorithm is required to design such an architecture, which also constrains the resulting learnt decoder. Here we remedy this requirement and propose a fully end-to-end trained neural decoder - *Deep Turbo Decoder* (DEEPTURBO). With novel learnable decoder structure and training methodology, DEEPTURBO reveals superior performance under both AWGN and non-AWGN settings as compared to the other two decoders - TURBO and NEURALBCJR. Furthermore, among all the three, DEEPTURBO exhibits the lowest error floor.

I. INTRODUCTION

A. Motivation

Communication standards typically fix an encoder as a capacity-approaching code, while allowing different stakeholders to implement their own decoders [23]. Thus, designing decoding algorithms with properties such as *high reliability*, *robustness*, *adaptivity* is of utmost interest to both industry and academia. Among several standard codes, turbo code is widely used in modern communication systems where the standard Turbo decoder (TURBO) uses the iterative BCJR [1] algorithm with an interleaving decoding procedure. Historically, this became the first channel coding scheme which achieves capacity-approaching performance under AWGN channels [10]. Despite the near-optimal performance on the AWGN channel, TURBO lacks both robustness and adaptivity on non-AWGN settings. Robustness refers to the upholding of the performance even at the face of unexpected noise. Since TURBO is an iterative decoding algorithm with interleaving, a log-likelihood corrupted by bursty noise on one coded bit can propagate to other coded bits via iterative decoding, which ends up with severely degraded decoding performance [36]. Adaptivity refers to the capability to adapt exhibiting competing performance over a number of different channels, when the channel statistics are known to the decoder. Traditional methods improve adaptivity via whitening and thresholding the received signal with heuristics, so as to make the AWGN-pre-designed decoder work well [23]. Heuristic algorithms show different levels of performance, but there is no guarantee on the performance under unexpected settings [24]. Moreover, Turbo's error floor refers to the flattened decoding performance on the high signal-to-noise ratio (SNR) region. As a result, Turbo code is not viable for applications which require high reliability such as secure communications and authentications [12] [14]. The existence of the Turbo error floor relies on the low-weight codeword distributions [14]. Multiple error floor lowering techniques have been proposed since the inception of Turbo code. Annealing methods transform the posterior of each stage to avoid transient chaos [17]. Designing a better interleaver to avoid low-weight codewords is proposed in [13]. Concatenating another outer code such as the Bose-Chaudhuri-Hocquenghem (BCH) code is also widely used [18]. Applying additional check code such as cyclic redundancy check (CRC) code for post-processing is proposed in [19] [20] [21]. Error floor lowering methods show different levels of success with handcrafted heuristics. In summary, the traditional design of TURBO with heuristics suffer to deliver on the desired features such as high reliability, robustness, adaptivity, and lower error floor.

B. Prior Art

Using deep learning based methods for channel coding has received sufficient attention since its inception [27] [28]. In particular, deep learning based decoders for canonical channel codes exhibit competing performance. BCH and High-Density Parity-Check (HDPC) codes can be decoded near optimally via a learnable Belief Propagation (BP) decoder [32] [33]. Polar codes can be also decoded by neural BP [34] [35]. Neural decoding for convolutional code and turbo code was introduced in [36], where in the case of turbo code, the introduced iterative neural BCJR decoder (NEURALBCJR) starts by imitating BCJR algorithm with RNN. It then equips the decoding with adaptivity by replacing BCJR algorithm with a pre-trained RNN followed by a further end-to-end fine turning of the parameters for non-AWGN settings. NEURALBCJR shows high reliability, with matching TURBO performance under different block lengths and signal-to-noise-ratio (SNR) on AWGN channel. On non-AWGN channels, NEURALBCJR is robust to unexpected noise, and can further be trained to adapt to other channel settings. NEURALBCJR imitates TURBO with RNN, thereby also inheriting its disadvantages, viz. the error floor. Furthermore, BCJR pre-training is required for training NEURALBCJR. A natural question to as here is : *Can we train a neural Turbo decoder without BCJR knowledge to get even better performance?*

C. Our Contribution

In this paper, we answer the above question in affirmative and introduce Deep Turbo Decoder (DEEPTURBO), the first end-to-end learnt capacity-approaching neural Turbo decoder without BCJR knowledge. DEEPTURBO has high reliability with respect to bit error rate (BER) and block error rate (BLER), robustness and adaptivity on various channels, and lower error floor as compared to TURBO and NEURALBCJR. DEEPTURBO also achieves desired BER performance with reduced decoding iterations. The paper is organized as follows:

Section II introduces the structure and training algorithm for DEEPTURBO, while Section III examines its performance on AWGN and non-AWGN channels. The paper concludes with discussion on the open issues and future directions in Section IV.

II. DEEPTURBO ARCHITECTURE

A. Turbo Encoder

Turbo encoder is composed of an interleaver and recursive systematic convolutional (RSC) encoders as shown in Figure 1, where the interleaver (π) shuffles the input string with a given order; while the deinterleaver (π^{-1}) undos the interleaving in a known order. RSC code with generating function $(1, \frac{f_1(x)}{f_2(x)})$ servers as the encoding block for Turbo code. Two commonly used configurations of RSC are used in this paper:

- code rate $R = 1/3$, with $f_1(x) = 1 + x^2$ and $f_2(x) = 1 + x + x^2$, which is denoted as turbo-757.

- code rate $R = 1/3$, with $f_1(x) = 1 + x^2 + x^3$ and $f_2(x) = 1 + x + x^3$, which is standard Turbo code used in LTE system, denoted as turbo-LTE.

Notation. A rate 1/3 turbo encoder generates three coded bits (x_1, x_2, x_3) per each message bit. As illustrated in Figure 1 (up), among the three coded bits, first output x_1 is the systematic bit, and x_2 is a coded bit generated through an RSC, and x_3 is a coded bit generated through an RSC for the interleaved bit stream. We let (y_1, y_2, y_3) denote the noise corrupted versions of (x_1, x_2, x_3) that the decoder receives.

B. Turbo Decoders

Turbo decoders are designed under the 'Turbo Principle' [16], which is iteratively refining posterior information by interleaved/de-interleaved received signals with soft-in soft-output (SISO) decoders. SISO decoder takes received signals and prior, and produces posterior as prior for later SISO blocks. The general Turbo decoder with rate 1/3 structure is shown in Figure 1 (down). Each decoding iteration uses two SISO decoders to decode, the first stage takes de-interleaved posterior p from last stage as prior, and received signal y_1 and y_2 as inputs; while the second stage takes interleaved posterior $\pi(q)$ as prior and received signal $\pi(y_1)$ and y_3. SISO outputs posterior q to be fed to the next stage. At the end of iterative decoding procedure, decoding is done according to the estimated posterior.
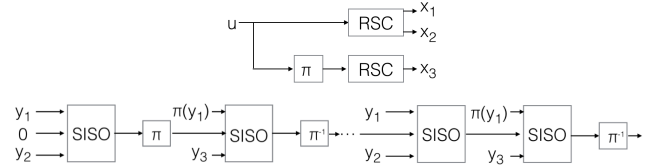

Fig. 1. Turbo Encoder (up) and Decoder (down)

C. Design of SISO Algorithm

Different Turbo decoding algorithms differ in the design of the SISO algorithm. We compare the SISO design of the TURBO, NEURALBCJR, and our proposed DEEPTURBO:

*1) Standard Turbo Decoder (*TURBO*):* TURBO [10] uses the BCJR algorithm for the SISO algorithm with extrinsic information and noise weighted systematic bits, cf. Figure 2 (left). The extrinsic information takes the difference between the prior and the posterior to be fed to the next stage. TURBO also requires estimating the channel noise variance to compensate noise. The posterior is compensated with extrinsic information and compensated systematic bits, before it is sent to next stage as prior. While TURBO is designed to operate reliably under AWGN settings, it is sensitive to non-AWGN noises as in a non-AWGN setting. For example, a bursty noise corrupted bit leads to severely degraded performance (shown in Figure 7), since the iterative decoding scheme propagates corrupted posterior to other code bits via interleaving. Even under AWGN channel, TURBO suffers from the error floor due to the existence of low weight codewords.

*2) Iterative Neural BCJR Decoder (*NEURALBCJR*) [36]:*
NEURALBCJR replaces the BCJR algorithm with Bidirectional Gated Recurrent Unit (Bi-GRU) (see appendix for more details). The SISO block of NEURALBCJR is shown in Figure 2 (right), where it is Initialized by pre-trained Bi-GRU with BCJR input and output to imitate BCJR algorithm followed by an end-to-end training till convergence. All SISO blocks share the same model weights. NEURALBCJR SISO block removes the link of compensating systematic bits and does not require estimating the channel noise to decode. NEURALBCJR avoids producing inaccurate noise weighted systematic bits by implicitly estimating the channel, which improves the robustness against unexpected non-AWGN channels. NEURALBCJR shows matched performance on AWGN channel compared to TURBO under AWGN channels, while it shows better robustness and adaptivity comparing to TURBO with heuristics. However, the error floor of NEURALBCJR is still high. In this design, there are two major caveats: (1) NEURALBCJR requires to have BCJR knowledge to initialize the SISO block. Without BCJR-initialization, directly training NEURALBCJR from scratch is not stable and (2) NEURALBCJR simply replaces the BCJR block by weight-sharing Bi-GRUs, with the same input and output relationship, which limits its potential capacity of NEURALBCJR.

*3) Deep Turbo Decoder (*DEEPTURBO*):* To ameliorate the caveats of NEURALBCJR, we propose DEEPTURBO where each SISO block (cf. Figure 2 (right)) still uses Bi-GRU as the building block, while keeping the extrinsic connection as a short-cut for gradient inspired by ResNet [2].

Two major structural differences between NEURALBCJR and DEEPTURBO:

- Non-shared weights. Unlike NEURALBCJR uses the same Bi-GRU for all SISO blocks, DEEPTURBO doesn't share weight across different iterations, which allows each iteration to deal with posterior differently. Furthermore, non-shared weights improve the training stability.
- More information passed to the next stage. Both TURBO and NEURALBCJR represent the posterior of each code bit by a single value log-likelihood (LLR). A single value for each code bit might not be sufficient to convey enough information. Inspired by resolving calibration issue by ensemble methods [5], for each code bit position, we take length K bits instead of 1 bit. For example, for block length $L = 100$, NEURALBCJR posterior LLR has shape $(L, 1)$, while DEEPTURBO transmits a posterior of shape (L, K) to next stage.

A significant advantage is that DEEPTURBO does not unitilize BCJR knowledge at all, which allows DEEPTURBO to learn a better decoding algorithm in a data-driven end-to-end approach. The hyper-parameters for DEEPTURBO decoder are shown in Figure 3, further discussion is deferred to the appendix.

III. DEEP TURBO DECODER PERFORMANCE

We compare the performance of Deep Turbo Decoder (DEEPTURBO) with the baseline decoders, the standardTurbo
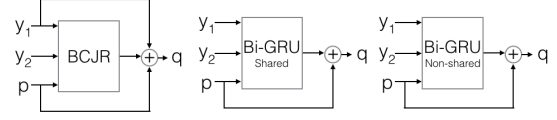

Fig. 2.   Different SISOs: TURBO (left), NEURALBCJR (middle) and DEEPTURBO (right)

DEEPTURBO SISO	2-layer Bi-GRU with 100 units
DEEPTURBO-CNN SISO	5-layer 1d CNN with 100 filters, kernel size = 5, stride=1, padding=2
Learning rate	0.001, decay by 10 when saturate
Num epoch	200
Block length	100/1000
Batch per epoch	100
Optimizer	Adam
Loss	BCE
Train SNR	-1.5dB
Batch size	500
Posterior feature size K	5
Decode iterations	6

Fig. 3.   DEEPTURBO hyperparameters

decoder (TURBO) and Iterative Neural BCJR Decoder (NEURALBCJR), under both AWGN and non-AWGN channels.
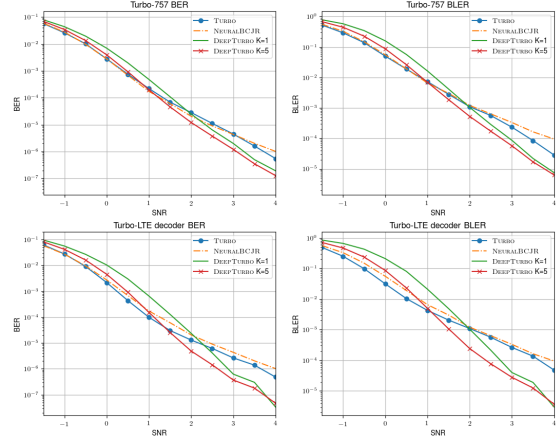

Fig. 4.   The proposed Deep Turbo Decoder (DEEPTURBO) improves upon the standard Turbo decoder in the large SNR regime for Turbo-757 (up) and Turbo-LTE (down)

A. AWGN with a block length 100

In Figure 4, we compare the decoder performances for Turbo codes with both turbo-757 and turbo-LTE, trained under block length 100, and decoding iteration 6. NEURALBCJR matches the performance of TURBO as expected. In all scenarios, DEEPTURBO outperforms both TURBO and NEURALBCJR on high SNR cases (SNR ≥ 0.5 dB), which implies lowered error floor. To achieve this performance, it is critical to use appropriate choice of the posterior information dimension K. Empirically we find $K = 5$ trains faster and also achieves the best performance among all $K < 10$.

In Figure 5, we compare the decoder performances of DEEPTURBO and TURBO with 2 and 6 decoding iterations. Compared to TURBO with 2 decoding iterations, DEEPTURBO ($i = 2$) shows significant improvement. This implies that the latent representations at lower layers (iterations) of DEEPTURBO extracts the information faster than iteratively

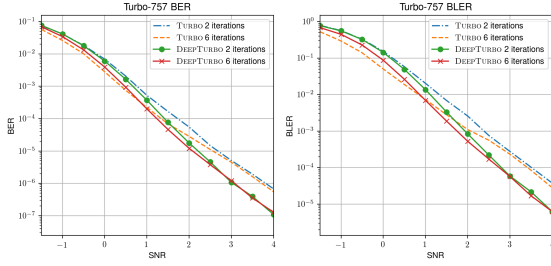applying BCJR. Hence, DEEPTURBO can achieve a desired level of accuracy with a smaller number of iterations.

Fig. 5. An intermediate layer of DEEPTURBO already achieves improved performance

B. Generalization to Longer Block Lengths

TURBO uses the BCJR algorithm, which is independent of the block length. Hence, TURBO is generalizable to any block lengths. On the other hand, the proposed DEEPTURBO trained on a short block length ($L = 100$) turbo code does not perform as well on a larger block length ($L = 1000$) turbo code when applied directly. This indicates that the gain of DEEPTURBO in the high SNR regime is due to customizing the decoder to the specific block length it is trained on. We can recover the desired performance on larger block length, by initializing DEEPTURBO with the model trained on shorter block lengths, and then further training it for small number of epochs. In Figure 6, we plot the BER and BLER of the DEEPTURBO after re-training under block length 1000.
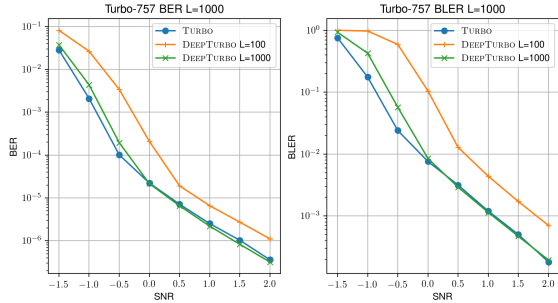

Fig. 6. Further training of DEEPTURBO is required to achieve the desired performance on larger block lengths: BER (left), and BLER (right)

Without relying on the mathematical structure of the code (as exploited in BCJR), generalizing to longer block lengths remains a challenging task. Ideally, we want a decoder that can be trained on short blocks which can be used on longer blocks. This will eliminate the bottleneck of several challenges in directly training longer blocks. For example, training under a large block length requires a large amount of GPU memory. Hence, we cannot train the decoder with large batch sizes, and this results in an unstable training. Furthermore, exploding and diminishing gradient of RNN makes learning unstable. It would be an interesting research direction to design good decoders that generalize to longer block lengths.

C. Non-AWGN performance

DEEPTURBO is tested on the following non-AWGN channels with block length 100:

- Additive T-distribution Noise (ATN) channel: $y = x + z$, where $z \sim T(\nu, \sigma^2)$.
- Radar Channel: $y = x + z + w$. where $z \sim N(0, \sigma_1^2)$ is a background AWGN noise, and $w \sim N(0, \sigma_2^2)$, with probability p is the radar noise with high variance and low probability. $\sigma_1 << \sigma_2$.

NEURALBCJR shows improved robustness and adaptivity compared to existing heuristics [36]. We train DEEPTURBO on ATN and Radar end-to-end. Figure 7 shows that DEEP-TURBO significantly improves upon NEURALBCJR. This is due to the non-shared parameters of DEEPTURBO, that can perform different decoding functions at different stages of decoding. Hence, DEEPTURBO has better adaptivity compared to NEURALBCJR.
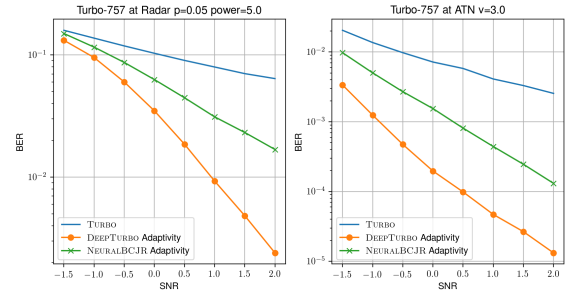

Fig. 7. DEEPTURBO adapts to non Gaussian channels: Radar channel with $p = 0.05$ and $\sigma_2 = 5.0$ (left), and ATN with $\nu = 3.0$ (right)

IV. CONCLUSION

We demonstrated that DEEPTURBO, an end-to-end trained decoder for turbo codes, exhibits an improved reliability, adaptivity and lowered error floor as compared to NEURAL-BCJR, while requiring no knowledge of a BCJR algorithm. It has a novel structure, which allows one to achieve required performance with reduced decoding iterations. We envision more end-to-end trained neural decoders such as DEEPTURBO will be proposed in the future for other state-of-the-art codes such as LDPC and Polar codes. We refer the reader to the appendix for further details.

REFERENCES

[1] Bahl L, Cocke J, Jelinek F, Raviv J. Optimal decoding of linear codes for minimizing symbol error rate (corresp.). IEEE Transactions on information theory. 1974 Mar;20(2):284-7.

[2] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 770-778).

[3] Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." Neural networks 2.5 (1989): 359-366.

[4] Schafer AM, Zimmermann HG. Recurrent neural networks are universal approximators. InInternational Conference on Artificial Neural Networks 2006 Sep 10 (pp. 632-640). Springer, Berlin, Heidelberg.

[5] Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles." Advances in Neural Information Processing Systems. 2017.

[6] Jiang Y, Kim H, Asnani H, Kannan S, Oh S, Viswanath P. LEARN Codes: Inventing Low-latency Codes via Recurrent Neural Networks. arXiv preprint arXiv:1811.12707. 2018 Nov 30.

[7] Tonnellier T, Leroux C, Le Gal B, Gadat B, Jego C, Van Wambeke N. Lowering the error floor of turbo codes with crc verification. IEEE Wireless Communications Letters. 2016 Aug;5(4):404-7.

[8] Shannon, C.E. "A mathematical theory of communication." Bell system technical journal 27.3 (1948): 379-423.

[9] Arikan, Erdal. "A performance comparison of polar codes and Reed-Muller codes." IEEE Communications Letters 12.6 (2008).

[10] Berrou, Claude, Alain Glavieux, and Punya Thitimajshima. "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1." Communications, 1993. ICC'93.

[11] MacKay, D. JC, and Radford M. N. "Near Shannon limit performance of low density parity check codes." Electronics letters 32.18 (1996).

[12] Richardson, Tom. "Error floors of LDPC codes." Proceedings of the annual Allerton conference on communication control and computing. Vol. 41. No. 3. The University; 1998, 2003.

[13] Boharquez RG, Nour CA, Douillard C. On the equivalence of interleavers for turbo codes. IEEE Wireless Communications Letters. 2015 Feb;4(1):58-61.

[14] Perez, Lance C., Jan Seghers, and Daniel J. Costello. "A distance spectrum interpretation of turbo codes." IEEE Transactions on information Theory 42.6 (1996): 1698-1709.

[15] Eidson, Donald Brian, Abraham Krieger, and Ramaswamy Murali. "Iterative decoder employing multiple external code error checks to lower the error floor." U.S. Patent No. 6,810,502. 26 Oct. 2004.

[16] Hagenauer, Joachim. "The turbo principle: Tutorial introduction and state of the art." Proc. International Symposium on Turbo Codes and Related Topics. 1997.

[17] Kocarev, L., Z. Tasev, and A. Vardy. "Improving turbo codes by control of transient chaos in turbo-decoding algorithms." Electronics Letters 38.20 (2002): 1184-1186.

[18] Andersen, Jakob Dahl. "Turbo codes extended with outer BCH code." Electronics letters 32.22 (1996): 2059-2060.

[19] Leanderson, Carl Fredrik, and C-EW Sundberg. "On list sequence turbo decoding." IEEE transactions on communications 53.5 (2005): 760-763.

[20] Ould-Cheikh-Mouhamedou, Youssouf, and Stewart Crozier. "Improving the error rate performance of turbo codes using the forced symbol method." IEEE communications letters 11.7 (2007).

[21] Tonnellier T, Leroux C, Le Gal B, Gadat B, Jego C, Van Wambeke N. Lowering the error floor of turbo codes with crc verification. IEEE Wireless Communications Letters. 2016 Aug;5(4):404-7.

[22] Duan L, Tsang IW, Xu D, Chua TS. Domain adaptation from multiple sources via auxiliary classifiers. InProceedings of the 26th Annual International Conference on Machine Learning 2009 Jun 14 (pp. 289-296). ACM.

[23] Richardson, Tom, and Ruediger Urbanke. Modern coding theory. Cambridge university press, 2008.

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

[25] Safavi-Naeini HA, Ghosh C, Visotsky E, Ratasuk R, Roy S. Impact and mitigation of narrow-band radar interference in down-link LTE. In2015 IEEE international conference on communications (ICC) 2015 Jun 8 (pp. 2644-2649). IEEE.

[26] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

[27] O'Shea, Timothy J., Kiran Karra, and T. Charles Clancy. "Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention." Signal Processing and Information Technology (IS-SPIT), 2016 IEEE International Symposium on. IEEE, 2016.

[28] O'Shea, Timothy, and Jakob Hoydis. "An introduction to deep learning for the physical layer." IEEE Transactions on Cognitive Communications and Networking 3.4 (2017): 563-575.

[29] Farsad, Nariman and Goldsmith, Andrea, "Neural Network Detection of Data Sequences in Communication Systems." IEEE Transactions on Signal Processing, Jan 2018.

[30] Dörner, Sebastian and Cammerer, Sebastian and Hoydis, Jakob and Brink, Stephan ten, "Deep learning-based communication over the air." arXiv preprint arXiv:1707.03384

[31] O'Shea, Timothy J., Tugba Erpek, and T. Charles Clancy. "Deep learning based MIMO communications." arXiv preprint arXiv:1707.07980 (2017).

[32] Nachmani, Eliya, Yair Be'ery, and David Burshtein. "Learning to decode linear codes using deep learning." Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference on. IEEE, 2016.

[33] Nachmani E, Marciano E, Lugosch L, Gross WJ, Burshtein D, Be'ery Y. Deep learning methods for improved decoding of linear codes. IEEE Journal of Selected Topics in Signal Processing. 2018 Feb;12(1):119-31.

[34] Gruber T, Cammerer S, Hoydis J, ten Brink S. On deep learning-based channel decoding. In2017 51st Annual Conference on Information Sciences and Systems (CISS) 2017 Mar 22 (pp. 1-6). IEEE.

[35] Cammerer S, Gruber T, Hoydis J, ten Brink S. Scaling deep learning-based decoding of polar codes via partitioning. InGLOBECOM 2017-2017 IEEE Global Communications Conference 2017 Dec 4 (pp. 1-6). IEEE.

[36] Kim, Hyeji and Jiang, Yihan and Rana, Ranvir and Kannan, Sreeram and Oh, Sewoong and Viswanath, Pramod. "Communication Algorithms via Deep Learning" Sixth International Conference on Learning Representations (ICLR), 2018.

[37] Hammer, Barbara, "On the approximation capability of recurrent neural networks." Neurocomputing Vol. 31, pp. 107–123, 2010.

[38] Aoudia, Faycal Ait, and Jakob Hoydis. "End-to-End Learning of Communications Systems Without a Channel Model." arXiv preprint arXiv:1804.02276 (2018).

[39] Felix A, Cammerer S, Dörner S, Hoydis J, Ten Brink S. Ofdm-autoencoder for end-to-end learning of communications systems. In2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC) 2018 Jun 25 (pp. 1-5). IEEE.

[40] Kim, Hyeji and Jiang, Yihan and Rana, Ranvir and Kannan, Sreeram and Oh, Sewoong and Viswanath, Pramod. "Deepcode: Feedback codes via deep learning." NeurIPS 2018.

[41] Viterbi, Andrew. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm." IEEE transactions on Information Theory 13.2 (1967): 260-269.

[42] Chung J, Gulcehre C, Cho K, Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555. 2014 Dec 11.

[43] Ovtcharov K, Ruwase O, Kim JY, Fowers J, Strauss K, Chung ES. Accelerating deep convolutional neural networks using specialized hardware. Microsoft Research Whitepaper. 2015 Feb 22;2(11):1-4.

[44] Sanders, Geoffrey A, "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.

[45] Kingma, D., and J. Ba Adam. "A method for stochastic optimization." International Conference on Learning Representations (ICLR). Vol. 5. 2015.

[46] Hwang K, Sung W. Single stream parallelization of generalized LSTM-like RNNs on a GPU. In2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2015 Apr 19 (pp. 1047-1051). IEEE.

[47] Seo M, Kembhavi A, Farhadi A, Hajishirzi H. Bidirectional attention flow for machine comprehension. Fifth International Conference on Learning Representations (ICLR), 2017.

APPENDIX

A. Deep Learning Modules

1) Recurrent Neural network (RNN) and Gated Recurrent Unit (GRU): RNN is one of the most widely used deep learning models for sequential data [26]. RNN can be considered as a learnable function $f(.)$ with input x_t and hidden state h_{t-1} at the time t, and outputs y_t for next layer and hidden state h_t for the next time. Wide enough RNN theoretically can approximate any function [4] [3]. Bidirectional RNN (Bi-RNN) [26], combines two RNNs, one for the forward pass and one for the backward pass. can use both information from the past and future. Bi-RNN can be considered as a generalization of forward-backward algorithm such as the

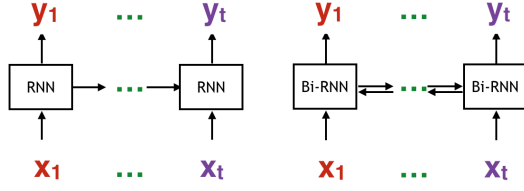BCJR algorithm [36]. RNN and Bi-RNN are shown in Figure 8.

Fig. 8.   Bi-RNN and RNN

RNN are not compatible to capture long-term dependencies due to exploding gradient problem, in practice Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) [26] are used to alleviate gradient exploding problem via gating schemes. GRU [42] is shown in Figure 9. This paper use Bidirectional GRU (Bi-GRU) as the primary learnable structure.
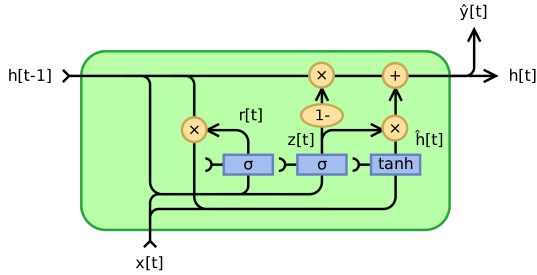

Fig. 9.   GRU

2) Same Shape 1D-CNN: 1D Convolutional Neural Network (1D-CNN) is widely used for sequential data with limited dependency length [26]. 1D-CNN takes the input of shape $(L, F1)$, with data block length L, and output feature size $F2$, and use $F2$ filters to conduct convolution with kernel size k. After convolving with kernel of odd size k, with each time the convolution kernel moves ahead by 1 step (termed as stride = 1) and the output length becomes $L - k + 1$.

Same shape 1D-CNN means that the input sequence and output sequence have the same length L. Same shape 1D-CNN has stride = 1, for each odd kernel size k, the zero padding is $\frac{k-1}{2}$ for both the beginning part and the ending part, such that the input and output block length L is preserved [26], as shown in Figure 10.
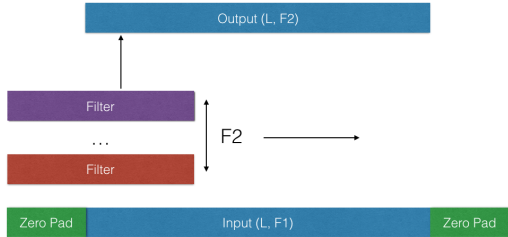

Fig. 10.   Same Shape 1D CNN with 1 layer

In this paper, we stack multiple same shape 1D-CNNs to construct a deep CNN based SISO decoder. Same shape

1D-CNN doesn't have a long dependency, we expect better trainability and slightly degraded performance. We use 5 layers, 100 filters same shape 1D-CNN, with kernel size 5. Empirically, the memory usage of 5 layer 100 filter same shape 1D-CNN is about 1/10 comparing to 2 layer 100 unit Bi-GRU, while the training time is about 1/5 of 2 layer 100 units Bi-GRU.

B. DEEPTURBO-CNN

RNNs are hard to parallelize for long blocks due to their sequential dependency structure [46]. RNN is also hard to train due to exploding gradient problem [42]. Replacing RNN with Convolutional Neural Network (CNN) has been proposed in recent NLP research [47]. Inspired by the success of CNNs for sequential data, we propose DEEPTURBO-CNN, which simply replaces the Bi-GRU of DEEPTURBO by multi-layer same shape 1D-CNN discussed in previous section. Since CNN has limited and fixed dependency length, we expect DEEPTURBO-CNN to have degraded performance comparing to DEEPTURBO. However, since CNN can be efficiently computed and trained with much less computation and memory, DEEPTURBO-CNN shows promise for future deployment.

We test the performance of DEEPTURBO-CNN with the same setup as Section III, with block length $L = 100$ and 6 decoding iterations. In Figure 11, DEEPTURBO-CNN exhibits a 0.5dB degradation compared to DEEPTURBO. DEEPTURBO-CNN requires much less computation and memory and is more efficient to run in parallel computing environments than DEEPTURBO.
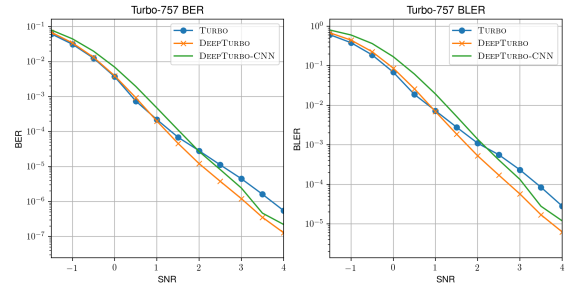

Fig. 11.   DEEPTURBO-CNN performance slightly degrades at the gain of computational efficiency on AWGN channel: BER (left) and BLER (right)

C. DEEPTURBO *Network Design and Training Methodology*

In this section, we discuss about the hyper-parameter design and training methodology of DEEPTURBO. We use either BER or learning curve to compare different designs of DEEPTURBO. The learning curve shows the validation loss dynamics during training, the x-axis is the training time measured in epoch, and the y-axis is the validation loss. The desired features are lower validation loss and faster convergence.

1) Optimal Information Feature Size K: DEEPTURBO performance is affected by information size K. Testing on different K is shown in Figure 12. Among different K, $K = 5$ shows the best performance. Smaller K limits the information transferred to the next stage, while larger K is harder to train.
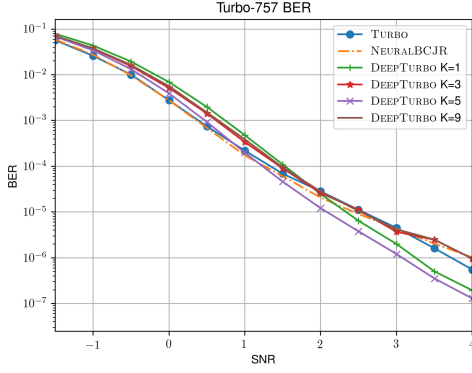
Fig. 12. $K = 5$ shows best performance among different selections

Fig. 14.   Learning curve of block length $L = 20$, larger network learns faster

2) Residual Connection improves training DEEPTURBO*:* Residual Neural Network (ResNet) [2] uses short-cut to improve the training of deep networks. Training DEEP-TURBO with residual connection not only imitates extrinsic information via existing Turbo decoder, but also improves training and shows a better performance, as shown in Figure 13.
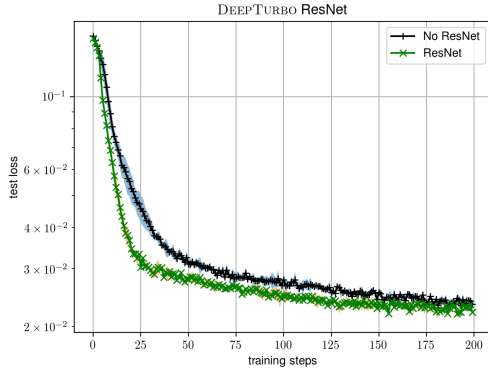

Fig. 15.   DEEPTURBO shows better trainability than NEURALBCJR without BCJR initialization

since adding Gaussian noise is equivalent to adding regularizer to the neural decoder [6]. Here we examine the effect of different training levels of SNR on DEEPTURBO. The learning curve is shown in Figure 16.

Training with 0dB shows fastest convergence, while training with -1.5dB shows slightly better validation loss when converge. So we schedule training SNR by starting to train at 0dB, and fine-tune at -1.5dB after convergence.

Fig. 13.   Improved training with ResNet

3) Larger network learns faster: Larger network with more units makes training faster, as shown in Figure 14. We choose 100 units due to computation limit. 100 units already show a better performance compared to NEURALBCJR, we expect even better performance with larger DEEPTURBO models depending upon the availability of computation resources.

4) DEEPTURBO *has better training stability than* NEU-RALBCJR*:* NEURALBCJR without BCJR initialization is very unstable during training, while DEEPTURBO shows progressive improvement as shown in Figure 15. Both NEURALBCJR and DEEPTURBO use Adam optimizer with learning rate 0.001.

5) Training SNR: Minimizing the Bit Error Rate (BER) of the neural decoder can be considered as a sequence of classification problems, where typically Binary Cross-Entropy (BCE) loss is used as a differentiable surrogate loss function [26]. Training decoder under low SNR close to Shannon Limit results in best empirical performance [36],
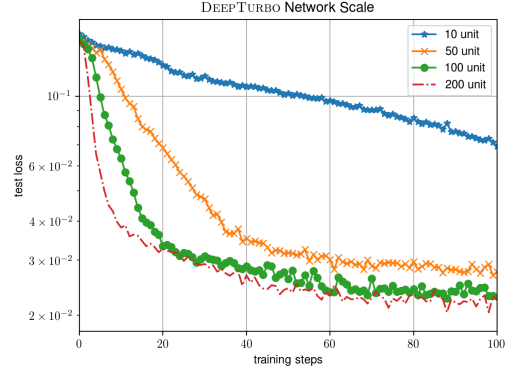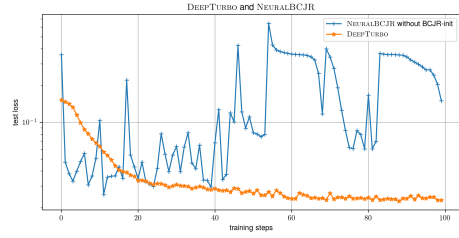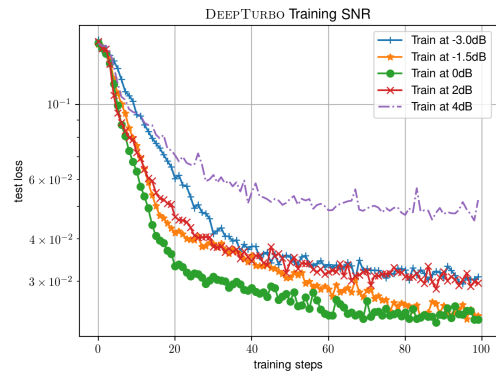

Fig. 16.   Optimal Training SNR

6) DEEPTURBO *Reduces Decoding Iterations:* Interpretation of the K dimensional latent representation is harder than interpreting log-likelihood (LLR) with dimension one. We examine the evolution of posteriors of each stage by auxiliary classifier method [22]. After DEEPTURBO converges, we fix the weights of all of its layers. For a given iteration

$i \in \{1, ..., 6\}$ with output posterior of shape (L, K), we add an auxiliary linear layer of shape $(K, 1)$ followed by sigmoid activation function to decode. We train auxiliary linear layer with posterior of shape (L, K) as input, and message $(L, 1)$ as output till convergence. Training the auxiliary linear layer harvests the information of refined posterior of dimension K. Figure 5 uses the auxiliary classifier method.

D. Open issues and future works

1) DEEPTURBO *Performance at Low SNR:* DEEP-TURBO's performance at low SNR is not as good as NEU-RALBCJR and the canonical Turbo decoder. The high SNR coding gain of DEEPTURBO is better, while at low SNR the coding gain is smaller. Designing neural decoder to work well for all levels of SNR is a direction for future research.

2) DEEPTURBO *Optimizing BLER Performance:* When the objective is to minimize Block Error Rate (BLER), training with BCE loss doesn't directly minimize the BLER. Using max-BCE loss minimizes the maximum BCE loss along with the whole sequence of length L: $max - BCE = \max_{i \in \{1, ..., L\}} BCE(f(x_i), y_i)$. As the max-BCE has sparser gradients, our empirical results show minimal gain for BLER. Improving BLER performance is an interesting future direction.

3) Design Better Learnable Structure: Although both turbo-LTE and turbo-757 performances are shown to have desired features, hyper-parameter tuning and training on turbo-LTE are experiencing much harder than that on turbo-757. Also training with longer block length is very time consuming. RNN has the inevitable gradient exploding problem, while CNN has limited and fixed dependency length. Designing a better learnable structure which improves both the capacity and the learnability is an interesting future research direction, which will have ramifications beyond channel coding community.

4) Model Reduction: DEEPTURBO has multiple iterations. Although non-shared weights across different decoding stages result in better performance, it is still favorable for the sake of reduced storage and deployment complexity to reduce the number of weights. One possible direction is to make part of model between each iteration share their parameters, thus can significantly reduce the number of weight to be saved for decoding model.

5) DEEPTURBO *Inspired Decoding Algorithm:* As DEEP-TURBO learns a neural decoder with better performance, one potential research direction is to distill what DEEPTURBO has learnt, and implement the algorithm to improve existing Turbo decoders.

6) Compare to State-of-the-art Error Floor Lowering Techniques: DEEPTURBO works well under short block lengths. However most error floor lowering results are on longer block length ($L \geq 528$). Comparing these results with existing state-of-the-art error floor lowering scheme can be an interesting and impactful future research direction.