Turbo Autoencoder: Deep learning based channel code for point-to-point communication channels

Anonymous Author(s)
Affiliation
Address
email

Abstract

Designing codes that combat the noise in a communication medium has remained a significant area of research in information theory as well as wireless communications. Asymptotically optimal channel codes have been developed by mathematicians for communication under canonical models after over 60 years of research. On the other hand, in many non-canonical channel settings, optimal codes do not exist and the codes designed for canonical models are adapted via heuristics to these channels and are thus not guaranteed to be optimal. In this work, we make significant progress on this problem by designing a fully end-to-end jointly trained neural encoder and decoder, namely, Turbo Autoencoder (TurboAE), with the following contributions: (a) under moderate block length, TurboAE approaches state-of-the-art performance under canonical channels; (b) moreover, TurboAE outperforms state-of-the-art codes under non-canonical settings in terms of reliability. TurboAE shows that the development of channel coding design can be automated via deep learning, with near-optimal performance.

1 Introduction

Autoencoder is a powerful unsupervised learning framework to learn latent representations by minimizing reconstruction loss of the input data [1]. Autoencoders have been widely used in unsupervised learning tasks such as representation learning [1] [2], denoising [3], and generative model [4] [5]. Most autoencoders are under-complete autoencoders, for which the latent space is smaller than the input data [2]. Over-complete autoencoders have latent space larger than input data. While the goal of under-complete autoencoder is to find a low dimensional representation of input data, the goal of over-complete autoencoder is to find a higher dimensional representation of input data so that from a noisy version of the higher dimensional representation, original data can be reliably recovered. Over-complete autoencoders are used in sparse representation learning [3] [6] and robust representation learning [7].

Channel coding aims at communicating a message over a noisy random channel [9]. As shown in Figure 1 left, the transmitter maps a message to a codeword via adding redundancy (this mapping is called encoding). A channel between the transmitter and the receiver randomly corrupts the codeword so that the receiver observes a noisy version which is used by the receiver to estimate the transmitted message (this process is called decoding). The encoder and the decoder together can be naturally viewed as an over-complete autoencoder, where the noisy channel in the middle corrupts the hidden representation (codeword). This coding and decoding process can thus be naturally modeled by over-complete autoencoders. Therefore, designing a reliable autoencoder can have a strong bearing on alternative ways of designing new encoding and decoding schemes for wireless communication systems.

Traditionally, communication algorithm design first involves designing a 'code' (i.e., the encoder) via optimizing certain mathematical properties of encoder such as minimum code distance [12]. The

associated decoder that minimizes the bit-error-rate is simply given by the maximum a posteriori (MAP) principle. While the optimal MAP decoder is computationally simple for some simple codes, such as convolutional codes, for known capacity-achieving codes, the MAP decoder is not computationally efficient and alternative decoding principles such as Turbo decoding and belief propagation are employed. The progress on the design of optimal channel codes with computationally efficient decoders has been quite sporadic due to its reliance on human ingenuity. Since Shannon's seminal work in 1948 [9], it took several decades of research to finally discover the current state-of-the-art codes [15].

Near-optimal channel codes such as Turbo [13], Low Density Parity Check (LDPC) [14], and Polar codes [15] show Shannon capacity-approaching [9] performance on AWGN channels, and they have had a tremendous impact on Long Term Evolution (LTE) and 5G standards. The traditional approach has the following caveats: (*a*) Decoder design relies on handcrafted optimal decoding algorithms on the canonical Additive White Gaussian Noise (AWGN) channel, where the signal is corrupted by i.i.d. Gaussian noise. When dealing with non-AWGN channels without closed-form representations, canonical codes with the layering of heuristics operate at best sub-optimality [12] [16]. (*b*) Channel coding transmits code block with K bits per message, which is referred as block length. Channel codes are guaranteed to be optimal only when the block-length approaches infinity, and thus are near-optimal in practice only when the block-length is large. Thus, there is a room for improvement under short and moderate block length regimes [17]. (*c*) The encoder designed for the AWGN channel is used across a large family of channels, while the decoder is adapted. This design methodology fails to utilize the flexibility of the encoder.

Related work. Deep learning has pushed the state-of-the-art performance of computer vision and natural language processing to a new level far beyond handcrafted algorithms in data-driven fashion [18]. There is a recent movement in applying deep learning to wireless communications. Deep learning based channel coding decoder design has been applied since [11] [19], where encoder is fixed as a near-optimal code. It is shown that belief propagation decoders for LDPC and Polar codes can be imitated by neural network [20] [21] [22] [23] [24]. It is also shown that convolutional and turbo codes can be decoded optimally via Recurrent Neural Network (RNN) [25] and Convolutional Neural Network (CNN) [26]. Equipping a decoder with a learnable neural network also allows fast adaptation via meta-learning [27]. Recent works also extend deep learning to multiple-input and multiple-output (MIMO) settings [28]. While *neural decoders* show improved performance on various communication channels, there has been limited success in inventing novel codes using this paradigm. Training methods for improving both modulation and channel coding are introduced in [11] [19], where a (7,4) neural code mapping a 4-bit message to a length-7 codeword can match (7,4) Hamming code performance. Current research includes training an encoder and a decoder with noisy feedback [10], improving modulation gain [29], as well as extensions to multi-terminal settings [30]. Joint source-channel coding shows improved results combining source coding (compression) along with channel coding (noise mitigation) [31]. Neural codes were shown to outperform existing state-of-the-art codes on the feedback channel [34]. However, in the canonical setting of AWGN channel, neural codes are still far from capacity-approaching performance due to the following challenges.

(Challenge A) Encoding with randomness is critical to harvest coding gain on long block length [9]. However, existing sequential neural models, both CNN and even RNN, can only learn limited local dependency [36]. Hence, neural encoder cannot sufficiently utilize the benefits of even moderate block length.

(Challenge B) Training neural encoder and decoder jointly (with a random channel in between) introduces optimization issues where the algorithm gets stuck at local optima. Hence, a novel training algorithm is needed.

Contributions. In this paper, we confront the above challenges by introducing Turbo Autoencoder (henceforth, TurboAE) – the first channel coding scheme with both encoder and decoder powered by neural networks that achieve reliability close to the state-of-the-art channel codes under AWGN channels for a moderate block length. We demonstrate that channel coding, which has been a focus of study by mathematicians for several decades [12], can be learned in an end-to-end fashion. Our major contributions are:

- We introduce TurboAE, a neural network based over-complete autoencoder parameterized as Convolutional Neural Networks (CNN) along with interleavers (permutation) and de-

interleavers (de-permutation) inspired by turbo codes (Section 3.1). We introduce TurboAE-binary, which binarizes the codewords via straight-through estimator (Section 3.2).

- We propose techniques that are critical for training TurboAE which includes mechanisms of alternate training of encoder and decoder as well as strategies to choose right training examples. Our training methodology ensures stable training of TurboAE without getting trapped at locally optimal encoder-decoder solutions. (Section 3.3)

- Compared to multiple capacity-approaching codes on AWGN channel, TurboAE shows superior performance in the low to middle SNR range when the block length is of moderate size ($K \sim 100$). To the best of our knowledge, this is the first result demonstrating the deep learning powered discovered neural codes can outperform traditional codes in the canonical AWGN setting (Section 4.1).

- On a non-AWGN channel, fine-tuned TurboAE shows significant improvements over state-of-the-art coding schemes due to the flexibility of encoder design, which shows that TurboAE has advantages on designing codes where handcrafted solutions fail (Section 4.2).

2   Problem Formation

The channel coding problem is illustrated in Figure 1 left, which consists of three blocks – an encoder $f_\theta(\cdot)$, a channel $c(\cdot)$, and a decoder $g_\phi(.)$. A channel $c(\cdot)$ randomly corrupts an input x and is represented as a probability transition function $p_{y|x}$. A canonical example of channel $c(\cdot)$ is an identically and independently distributed (i.i.d.) AWGN channel, which generates $y_i = x_i + z_i$ for $z_i \sim N(0, \sigma^2)$, $i = 1, \cdots, K$. The encoder $x = f_\theta(u)$ maps a random binary message sequence $u = (u_1, \cdots, u_K) \in \{0,1\}^K$ of block length K to a codeword $x = (x_1, \cdots, x_N)$ of length N, where x must satisfy either soft power constraint where $E(x) = 0$ and $E(x^2) = 1$, or hard power constraint $x \in \{-1, +1\}$. Code rate is defined as $R = \frac{K}{N}$, where $N > K$. The decoder $g_\phi(y)$ maps a real valued received sequence $y = (y_1, \cdots, y_N) \in \mathcal{R}^N$ to an estimate of the transmitted message sequence $\hat{u} = (\hat{u}_1, \cdots, \hat{u}_K) \in \{0,1\}^K$.

AWGN channel allows closed-form mathematical analysis, which has remained as the major playground for channel coding researchers. The noise level is defined as signal-to-noise ratio, $SNR = -10 \log_{10} \sigma^2$. The decoder recovers the original message as $\hat{u} = g_\phi(y)$ using the received signal y.

Channel coding aims to minimize the error rate of recovered message $\hat{u}$. The standard metrics are bit error rate (BER), defined as $BER = \frac{1}{K} \sum_1^K \Pr(\hat{u}_i \neq u_i)$, and block error rate (BLER), defined as $BLER = \Pr(\hat{u} \neq u)$.

While canonical capacity-approaching channel codes work well as block length goes to infinity, when the block length is short, they are not guaranteed to be optimal. We show the benchmarks on block length 100 in Figure 1 right with widely-used LDPC, Turbo, Polar, and Tail-bitting Convolutional Code (TBCC), generated via Vienna 5G simulator [37] [38], with code rate $1/3$.
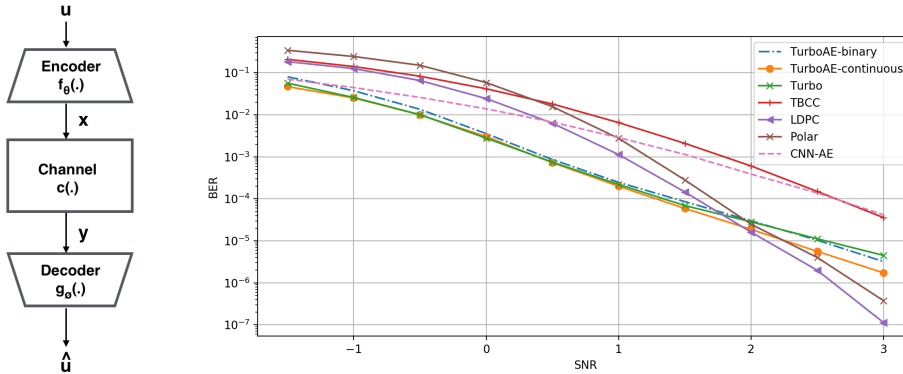

Figure 1: Channel coding can be viewed as an over-complete autoencoder with channel in the middle (left). TurboAE performs well under moderate block length in low and middle SNR (right).

Naively applying deep learning models by replacing encoder and decoder with general purpose neural network does not perform well. Direct applications of fully connected neural network (FCNN) cannot scale to a longer block length; the performance of FCNN-AE is even worse than repetition code [35]. Direct applications where both the encoder and the decoder are Convolutional Autoencoder (termed as CNN-AE [39]) shows better performance than TBCC, but are far from capacity-approaching codes such as LDPC, Polar, and Turbo. Bidirectional RNN and LSTM [35] has similar performance as CNN-AE and is not shown in the figure for clarity. Thus neither CNN nor RNN based auto-encoders can directly approach state-of-the-art performance. A key reason for their shortcoming is that they have only local memory, the encoder only remembers information locally. To have high protection against channel noise, it is necessary to have long term memory.

We propose TurboAE with interleaved encoding and iterative decoding that creates long term memory in the code and shows a significant improvement compared to CNN-AE. TurboAE has two versions, TurboAE-continuous which faces soft power constraint (i.e., the total power across a codeword is bounded) and TurboAE-binary which faces hard power constraint (i.e., each transmitted symbol has a power constraint - and is thus forced to be binary). Both TurboAE-binary and TurboAE-continuous perform comparable or better than all other capacity-approaching codes at a low SNR, while at a high SNR (over 2 dB with $BER < 10^{-5}$), the performance is only worse than LDPC and Polar code.

3 TurboAE : Architecture Design and Training

3.1 Design of TurboAE

Turbo code and turbo principle: Turbo code is the first capacity-approaching code ever designed [13]. There are two novel components of Turbo code which led to its success: an interleaved encoder and an iterative decoder. The starting point of the Turbo code is a recursive systematic convolutional (RSC) code which has an optimal decoding algorithm (the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [41]). A key disadvantage in the RSC code is that the algorithm lacks long range memory (since the convolutional code operates on a sliding window). The key insight of Berrou was to introduce long range memory by creating two copies of the input bits - the first goes through the RSC code and the second copy goes through an interleaver (which is a permutation of the bits) before going through the same code. Such a code can be decoded by iteratively alternating between soft-decoding based on the signal received from the first copy and then using the de-interleaved version as a prior to decode the second copy. The 'Turbo principle' [40] refers to the iterative decoding with successively refining the posterior distribution on the transmitted bits across decoding stages with original and interleaved order. This code is known to have excellent performance, and inspired this, we design TurboAE featuring both learnable interleaved encoder and iterative decoder.

Interleaved Encoding Structure: Interleaving is widely used in communication systems to mitigate bursty noise [42]. Formally, interleaver $x^\pi = \pi(x)$ and de-interleaver $x = \pi^{-1}(x^\pi)$ shuffle and shuffle back the input sequence x with the a pseudo random interleaving array known to both encoder and decoder, respectively, as shown in Figure 2 left. In the context of Turbo code and TurboAE, the interleaving is not used to mitigate bursty errors (since we are mainly concerned with i.i.d. channels) but rather to add long range memory in the structure of the code.

We take code rate 1/3 as an example for interleaved encoder f_θ, which consists of three learnable encoding blocks $f_{i,\theta}(.)$ for $i \in \{1, 2, 3\}$, where $f_{i,\theta}(.)$ encodes $b_i = f_\theta(u), i \in \{1, 2\}$ and $b_3 = f_{3,\theta}(\pi(u))$, where b_i is a continuous value. The power constraint of channel coding is enforced via power constraint block $x_i = h(b_i)$.
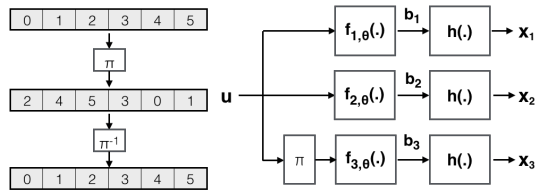

Figure 2: Visualization of Interleaver (π) and De-interleaver (π^{-1}) (left); TurboAE encoder on code rate 1/3 (right)

4

172 **Iterative Decoding Structure**: As received codewords are encoded from original message u and
173 interleaved message $\pi(u)$, decoding interleaved code requires iterative decoding on both interleaved
174 and de-interleaved order shown in Figure 3. Let y_1, y_2, y_3 denote noisy versions of x_1, x_2, x_3,
175 respectively. The decoder runs multiple iterations, with each iteration contains two decoders $g_{\phi_{i,1}}$
176 and $g_{\phi_{i,2}}$ for interleaved and de-interleaved order on the i-th iteration.

177 The first decoder $g_{\phi_{i,1}}$ takes received signal y_1, y_2 and de-interleaved prior p with shape (K, F),
178 where as F is the information feature size for each code bit, to produce the posterior q with same
179 shape (K, F). The second decoder $g_{\phi_{i,2}}$ takes interleaved signal $\pi(y_1)$, y_3 and interleaved prior p to
180 produce posterior q. The posterior of previous stage q serves as the prior of next stage p. The first
181 iteration takes 0 as a prior, and at last iteration the posterior is of shape $(K, 1)$, are decoded as by
182 sigmoid function as $\hat{u} = sigmoid(q)$.

183 Both encoder and decoder structure can be considered as a parametrization of Turbo code. Once we
184 parametrize the encoder and the decoder, since the encoder, channel, and decoder are differentiable,
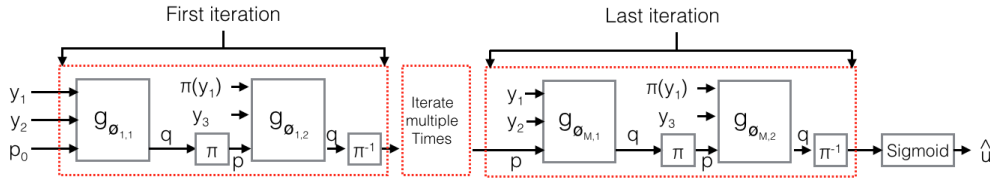185 TurboAE can be trained end-to-end via gradient descent and its variants.

Figure 3: TurboAE iterative decoder on code rate 1/3

186 **Encoder and Decoder Design**: The space of messages and codewords are exponential (For a length-
187 K binary sequence, there are 2^K distinct messages). Hence, the encoder and decoder must have some
188 structural restrictions to ensure generalization to messages unseen during the training [8]. Applying
189 parameter-sharing sequential neural models such as CNN and RNN are natural parametrization
190 methods for both the encoding and the decoding blocks.

191 RNN models such as Gated Recurrent Unit (GRU) and Long-Short Term Memory (LSTM) are
192 commonly used for sequential modeling problems [43]. RNN is widely used in deep learning based
193 communications systems [25] [34] [26] [35], as RNN has a natural connection to sequential encoding
194 and decoding algorithms such as convolutional code and BCJR algorithm [25].

195 However RNN models are: (1) of higher complexity than CNN models, (2) harder to train due to
196 gradient explosion, and (3) harder to run in parallel [36]. In this paper, we use one dimensional CNN
197 (1D-CNN) as the alternative encoding and decoding model. Although the longest dependency length
198 is fixed, 1D-CNN has lower complexity, better trainability [44], and easier implementation in parallel
199 via AI-chips [45]. The learning curve comparison between CNN and RNN is shown in Figure 4 left.
200 Training CNN-based model converges faster and more stable than RNN-based GRU model.

201 **Power Constraint Block**: The operation of power constraint blocks (i.e., $h(\cdot)$ in $x = h(b)$) depends
202 on the requirement of power constraint.

203 Soft power constraint normalize the power of code, as $E(x) = 0$ and $E(x^2) = 1$. TurboAE-
204 continuous with soft power constraint allows the code x to be continuous. Addressing the statistical
205 estimation issue given a limited batch size, we use normalization method [46] as:$x_i = \frac{b_i - \mu(b)}{\sigma(b)}$,
206 where $\mu(b) = \frac{1}{K} \sum_{i=1}^{K} b_i$ and $\sigma(b) = \frac{1}{K} \sum_{i=1}^{K} (b_i - \mu(b))^2$ are scalar mean and standard deviation
207 estimation of the whole block. During the training phase, $\mu(b)$ and $\sigma(b)$ are estimated from the whole
208 batch. On the other hand, during the testing phase, $\mu(b)$ and $\sigma(b)$ are pre-computed with multiple
209 batches. The normalization layer can be also considered as BatchNorm without affine projection,
210 which is critical to stabilize the training of the encoder [47].

3.2 Design of TurboAE-binary – Binarization via Straight-Through Estimator

212 Some wireless communication system requires hard power constraint, where the encoder output is
213 binary as $x \in \{-1, +1\}$ [48] - so that every symbol has exactly the same power and the information
214 is conveyed in the sign. Hard power constraint is not differentiable, since restricting $x \in \{-1, +1\}$

215 via $x = \text{sign}(b)$ has zero gradient almost everywhere. We combine normalization and Straight-
216 Through Estimator (STE) [49] [50] to bypass this differentiability issue. STE passes the gradient of
217 $x = \text{sign}(b)$ as $\frac{\partial x}{\partial b} = \mathbb{1}(|b| \leq 1)$, enables training encoder by passing estimated gradients to encoder,
218 while enforcing hard power constraint.

219 Simply training with STE cannot learn a good encoder as shown in Figure 4 right. To mitigate the
220 trainability issue, we apply pre-training, which pre-trains TurboAE-continuous firstly, and then add
221 the hard power constraint on top of soft power constraint as $x = \text{sign}(\frac{b-\mu(b)}{\sigma(b)})$, whereas the gradient
222 is estimated via STE. Figure 4 right shows that with pre-training, TurboAE-binary reaches Turbo
223 performance within 100 epochs of fine-tuning.

224 TurboAE-binary is slightly worse than TurboAE-continuous as shown in Figure 1, especially at high
225 SNR, since: (a) TurboAE-continuous can be considered as a joint coding and high order modulation
226 scheme, which has a larger capacity than binary coding at high SNR [48], and (b) STE is an estimated
227 gradient, which makes training encoder more noisy and less stable.
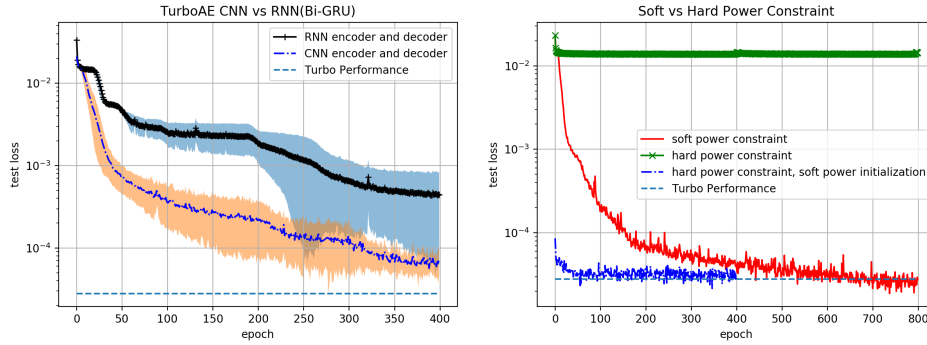

Figure 4: Learning Curves on CNN vs GRU: CNN shows faster training convergence (left); Training
with STE requires soft-constraint pre-training (right)

3.3 Neural Trainability Design

229 The training algorithms for training TurboAE are shown in Algorithm 1. Compared to the conventional
230 deep learning model training, training TurboAE has the following differences:

Algorithm 1 Training Algorithm for TurboAE

Require: Batch Size B, Train Encoder Steps T_{enc}, Train Decoder Steps T_{dec}, Number of Epoch M
    Encoder Training SNR σ_{enc}, Decoder Training SNR σ_{dec}
    for $i \leq M$ **do**
        for $j \leq T_{enc}$ **do**
            Generate random training example u, and random noise $z \sim N(0, \sigma_{enc})$.
            Train encoder f_θ with decoder fixed, with u and z.
        end for
        for $j \leq T_{dec}$ **do**
            Generate random training example u, and random noise $z \sim N(0, \sigma_{dec})$.
            Train decoder g_ϕ with encoder fixed, with u and z.
        end for
    end for

---

231 • **Very Large Batch Size** Large batch size is critical to average the channel noise effects.
232   Empirically, TurboAE reaches Turbo performance only when the batch size is grater than
233   500.

234 • **Train Encoder and Decoder Separately** We train encoder and decoder separately as shown
235   in Algorithm 1, to avoid getting stuck in local optimum [10] [35].

- **Different Training Noise Level for Encoder and Decoder** Empirically, while it is best to train a decoder at a low training SNR as discussed in [25], it is best to train an encoder at a training SNR that matches the testing SNR, e.g training encoder at 2dB results in good encoder when testing at 2dB [35]. In this work, we use random selection of -1.5 to 2 dB for training the decoder, and test and train the encoder at the same SNR.

We do a detailed analysis of training algorithm in the supplementary materials. The hyper-parameters are shown in Table 1.

Loss	Binary Cross-Entropy (BCE)
Encoder	2 layers 1D-CNN, kernel size 5, 100 filters for each $f_{i,\theta}(.)$ block
Decoder	5 layers 1D-CNN, kernel size 5, 100 filters for each $g_{\phi_{i,j}(.)}$ block
Decoder Iterations	6
Info Feature Size F	5
Batch Size	500 when start, double when saturates for 20 epochs, till reaches 2000
Optimizer	Adam with initial learning rate 0.0001
Training Schedule for Each Epoch	Train encoder $T_{enc} = 100$ times, then train decoder $T_{dec} = 500$ times
Block Length K	100
Number of Epochs M	800

Table 1: Hyper-parameters of TurboAE

4 Experiment Results

4.1 Block length coding gain of TurboAE

As block length increases, better reliability can be achieved via channel coding, which is referred to as *coding gain* [13]. We compare TurboAE (only TurboAE-continuous is shown in this section) with the Turbo code and CNN-AE, tested at BER at 2dB on different block lengths, shown in Figure 5 left. Both CNN-AE and TurboAE are trained with block length 100, and tested on other block lengths. As block length increases, CNN-AE shows saturating coding gain, while TurboAE and Turbo code reduce the error rate as the block length increases. Naively applying general purpose neural network such as CNN to channel coding problem cannot gain performance on long block lengths.

Note that TurboAE is still worse than Turbo when the block length is large, since long block length requires large memory usage and more complicated structure to train. Improving TurboAE on very long block length remains open as an interesting future direction.

The BER performance boosted by neural architecture design is shown in Figure 5 right. We compare the fine-tuned performance among CNN-AE, TurboAE, and TurboAE without interleaving as $x^\pi = \pi(x)$. TurboAE with interleaving significantly outperforms TurboAE without interleaving and CNN-AE.
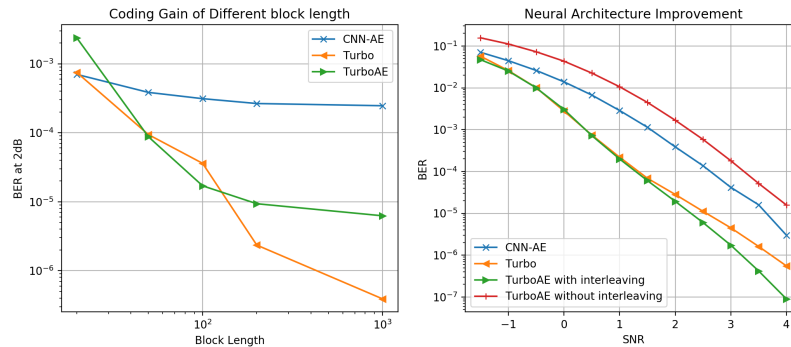

Figure 5: Interleaving improves coding gain (left); Neural Architecture improves BER performance (right).

4.2 Performance on non-AWGN channels

Typically there are no close-form solutions under non-AWGN and non-iid channels. We compare two benchmarks: (1) canonical Turbo code, and (2) DeepTurbo Decoder [26], a neural decoder fine-tuned at the given channel. We test the performance of both iid channels and non-iid channels in settings which are as:

(a) iid Additive T-distribution Noise (ATN) Channel, with $y_i = x_i + z_i$, where iid $z_i \sim T(\nu, \sigma^2)$ is heavy-tail (tail weight controlled based on the parameter $\nu = 3.0$) T-distribution noise with variance σ^2. The performance is shown in Figure 6 left.

(b) non-iid Markovian-AWGN channel, is a special AWGN channel with good and bad states. At bad state the noise is worse by 1dB than the SNR, and at good state, the noise is better by 1dB than the SNR. The state transition probability between good and bad states are symmetric as $p_{bg} = p_{gb} = 0.8$. The performance is shown in Figure 6 right.

For both ATN and Markovian-AWGN channels, DeepTurbo outperforms canonical Turbo code. TurboAE-continuous with learnable encoder, outperform DeepTurbo in both the cases. TurboAE-binary outperforms DeepTurbo on ATN channel, while on Markovian-AWGN channel, TurboAE-binary while not performing better than DeepTurbo at high SNR regimes, still outperforms canonical Turbo. With the flexibility of designing encoder, TurboAE designs better code than handcrafted Turbo code, for channels without a close-form mathematical representation.
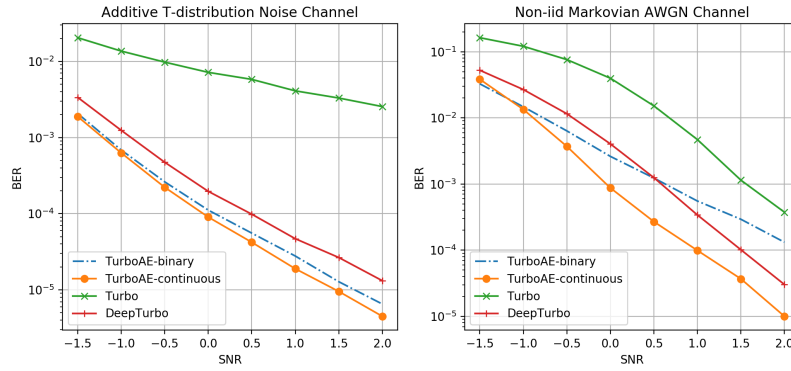

Figure 6: TurboAE on iid ATN channel (left) and on-iid Markovian-AWGN channel (right)

5 Conclusion

In this paper, we propose TurboAE, an end-to-end learnt channel coding scheme with novel neural structure and training algorithms. TurboAE learns capacity-approaching code on various channels under moderate block length by building upon 'turbo principle' and thus, exhibits discovery of codes for channels where a closed-form representation may not exist. TurboAE, hence, brings an interesting research direction to design channel coding algorithms via joint encoder and decoder design.

References

[1] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning.* ACM, 2008, pp. 1096–1103.

[2] A. Krizhevsky and G. E. Hinton, "Using very deep autoencoders for content-based image retrieval." in *ESANN*, 2011.

[3] A. Makhzani and B. Frey, "K-sparse autoencoders," *arXiv preprint arXiv:1312.5663*, 2013.

[4] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Advances in neural information processing systems*, 2016, pp. 2172–2180.

[5] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[6] L. Deng, M. L. Seltzer, D. Yu, A. Acero, A.-r. Mohamed, and G. Hinton, "Binary coding of speech spectrograms using a deep auto-encoder," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[7] Q. V. Le, A. Karpenko, J. Ngiam, and A. Y. Ng, "Ica with reconstruction cost for efficient overcomplete feature learning," in *Advances in neural information processing systems*, 2011, pp. 1017–1025.

[8] S. Dörner, S. Cammerer, J. Hoydis, and S. t. Brink, "Deep learning-based communication over the air," *arXiv preprint arXiv:1707.03384*, 2017.

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

[10] F. A. Aoudia and J. Hoydis, "End-to-end learning of communications systems without a channel model," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2018, pp. 298–303.

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

[12] T. Richardson and R. Urbanke, *Modern coding theory*. Cambridge university press, 2008.

[13] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proceedings of ICC'93-IEEE International Conference on Communications*, vol. 2. IEEE, 1993, pp. 1064–1070.

[14] D. J. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics letters*, vol. 33, no. 6, pp. 457–458, 1997.

[15] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes," in *2008 IEEE International Symposium on Information Theory*. IEEE, 2008, pp. 1173–1177.

[16] J. Li, X. Wu, and R. Laroia, *OFDMA mobile broadband communications: A systems approach*. Cambridge University Press, 2013.

[17] Y. Polyanskiy, H. V. Poor, and S. Verdú, "Channel coding rate in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 56, no. 5, p. 2307, 2010.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[19] T. J. O'Shea and J. Hoydis, "An introduction to machine learning communications systems," *arXiv preprint arXiv:1702.00832*, 2017.

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

[21] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be'ery, "Deep learning methods for improved decoding of linear codes," *IEEE Journal of Selected Topics in Signal Processing*, 2018.

[22] W. Xu, Z. Wu, Y.-L. Ueng, X. You, and C. Zhang, "Improved polar decoder based on deep learning," in *2017 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2017, pp. 1–6.

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

[24] S. Cammerer, T. Gruber, J. Hoydis, and S. ten Brink, "Scaling deep learning-based decoding of polar codes via partitioning," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

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

[26] Y. Jiang, H. Kim, S. Kannan, S. Oh, and P. Viswanath, "Deepturbo: Deep turbo decoder," 2019.

[27] Y. Jiang, H. Kim, H. Asnani, and S. Kannan, "Mind: Model independent neural decoder," *arXiv preprint arXiv:1903.02268*, 2019.

[28] N. Samuel, T. Diskin, and A. Wiesel, "Deep mimo detection," in *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2017, pp. 1–5.

[29] A. Felix, S. Cammerer, S. Dörner, J. Hoydis, and S. t. Brink, "Ofdm-autoencoder for end-to-end learning of communications systems," *arXiv preprint arXiv:1803.05815*, 2018.

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

[31] K. Choi, K. Tatwawadi, T. Weissman, and S. Ermon, "Necst: Neural joint source-channel coding," *arXiv preprint arXiv:1811.07557*, 2018.

[32] L. Huang, H. Zhang, R. Li, Y. Ge, and J. Wang, "Ai coding: Learning to construct error correction codes," *arXiv preprint arXiv:1901.05719*, 2019.

[33] ——, "Reinforcement learning for nested polar code construction," *arXiv preprint arXiv:1904.07511*, 2019.

[34] H. Kim, Y. Jiang, S. Kannan, S. Oh, and P. Viswanath, "Deepcode: Feedback codes via deep learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 9436–9446.

[35] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "Learn codes: Inventing low-latency codes via recurrent neural networks," *arXiv preprint arXiv:1811.12707*, 2018.

[36] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[37] M. K. Muller, F. Ademaj, T. Dittrich, A. Fastenbauer, B. R. Elbal, A. Nabavi, L. Nagel, S. Schwarz, and M. Rupp, "Flexible multi-node simulation of cellular mobile communications: the Vienna 5G System Level Simulator," *EURASIP Journal on Wireless Communications and Networking*, vol. 2018, no. 1, p. 17, Sep. 2018.

[38] B. Tahir, S. Schwarz, and M. Rupp, "Ber comparison between convolutional, turbo, ldpc, and polar codes," in *2017 24th International Conference on Telecommunications (ICT)*. IEEE, 2017, pp. 1–7.

[39] B. Zhu, J. Wang, L. He, and J. Song, "Joint transceiver optimization for wireless communication phy with convolutional neural network," *arXiv preprint arXiv:1808.03242*, 2018.

[40] J. Hagenauer, "The turbo principle: Tutorial introduction and state of the art," in *Proc. International Symposium on Turbo Codes and Related Topics*, 1997, pp. 1–11.

[41] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Transactions on information theory*, vol. 20, no. 2, pp. 284–287, 1974.

[42] H. R. Sadjadpour, N. J. Sloane, M. Salehi, and G. Nebe, "Interleaver design for turbo codes," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 5, pp. 831–837, 2001.

[43] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[44] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of cnn and rnn for natural language processing," *arXiv preprint arXiv:1702.01923*, 2017.

[45] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," *Microsoft Research Whitepaper*, vol. 2, no. 11, pp. 1–4, 2015.

[46] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[47] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" in *Advances in Neural Information Processing Systems*, 2018, pp. 2483–2493.

[48] D. Tse and P. Viswanath, *Fundamentals of wireless communication*. Cambridge university press, 2005.

[49] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[50] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in neural information processing systems*, 2016, pp. 4107–4115.