An Iterative BP-CNN Architecture for Channel Decoding

Fei Liang, Cong Shen, *Senior Member, IEEE*, and Feng Wu, *Fellow, IEEE*

***Abstract*—Inspired by the recent advances in deep learning, we propose a novel iterative belief propagation – convolutional neural network (BP-CNN) architecture for channel decoding under correlated noise. This architecture concatenates a trained CNN with a standard BP decoder. The standard BP decoder is used to estimate the coded bits, followed by a CNN to remove the estimation errors of the BP decoder and obtain a more accurate estimation of the channel noise. Iterating between BP and CNN will gradually improve the decoding SNR and, hence, result in better decoding performance. To train a well-behaved CNN model, we define a new loss function that involves not only the accuracy of the noise estimation but also the normality test for the estimation errors, i.e., to measure how likely the estimation errors follow a Gaussian distribution. The introduction of the normality test to the CNN training shapes the residual noise distribution and further reduces the bit error rate of the iterative decoding, compared to using the standard quadratic loss function. We carry out extensive experiments to analyze and verify the proposed framework.[1]**

***Index Terms*—Belief propagation, channel coding, neural networks.**

I. Introduction

CHANNEL encoding and decoding are important components in modern communication systems and tremendous progresses have been made both in coding theory and its applications. For example, low-density parity-check (LDPC) codes [1] are able to yield a performance close to the Shannon capacity for AWGN channels with a properly optimized encoding structure and the well developed belief propagation (BP) decoding algorithm [2]. However, in practical communication systems, channels may exhibit correlations in fading [3], [4] and noise. We focus on the noise correlation in this paper which can happen because of filtering, oversampling [5], and device noise [6]. For example, in power line communication, colored noise is often caused by common buildings and residential electronic equipments [7]. Another example is that in 10GBASE-t Ethernet, various filtering schemes are needed to combat the effect of inter-symbol interference, which causes correlated noise [8]. In addition, correlation can be found in device noise of digital systems such as pink noise caused by phase noise and clock jitter [6], and magnetic recording [9].

A well-designed channel code may not have satisfactory performance if the receiver is not designed to handle noise correlation [10]. Specifically, LDPC codes will encounter a performance degradation under colored noise [8]. The difficulty in addressing this issue mainly comes from the high complexity introduced by the colored noise. The most straightforward method to address this issue is *whitening*, i.e. to transform colored noise to white noise. However, this method needs matrix multiplication, which is of high complexity for long-length codes. Furthermore, the equivalent coded symbols after whitening may exhibit a different structure than the transmitted symbols, which complicates the decoding. In theory, the decoder can first estimate the noise distribution, and then optimize the BP decoder using the estimated joint distribution. This approach may have very high complexity when the correlation is strong. Hence, a low-complexity and robust decoder architecture that can exploit the characteristics of noise correlation without relying on its specific structure is desired.

Recent advances in deep learning provide a new direction to tackle this problem. Instead of deriving an algorithm from a pre-defined channel model, deep learning technologies allow the system to learn an efficient network model directly from training data. Deep learning has been applied in computer vision [11], natural language processing [12], autonomous vehicles [13] and many other areas, and the results have been quite remarkable. Inspired by these advances, researchers have recently tried to solve communication problems (including channel decoding) using deep learning technologies [14]–[24], and a summary of these works is provided in Section II. However, none of these works address the problem of efficient decoding of linear codes under correlated channel noise.

In this paper, we design a novel receiver architecture to tackle the decoding problem when correlation exists in channel noise. This architecture concatenates a trained convolutional neural network (CNN) with a standard BP decoder, and the received symbols are iteratively processed between BP and CNN – hence the name *iterative BP-CNN*. At the receiver side, the received symbols are first processed by the BP decoder to obtain initial decoding results. Then, subtracting the estimated transmit symbols from the received symbols, we obtain an estimation of

Manuscript received July 14, 2017; revised November 13, 2017 and January 7, 2018; accepted January 9, 2018. Date of publication January 15, 2018; date of current version February 16, 2018. This work was supported by the National Natural Science Foundation of China under Grants 61631017 and 61572455. The guest editor coordinating the review of this paper and approving it for publication was Dr. Silvija Kokalj-Filipovic. *(Corresponding author: Cong Shen.)*

The authors are with the Laboratory of Future Networks, School of Information Science and Technology, University of Science and Technology of China, Hefei 230027, China (e-mail: lfbeyond@mail.ustc.edu.cn; congshen@ustc.edu.cn; fengwu@ustc.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSTSP.2018.2794062

[1]Code is available at https://github.com/liangfei-info/Iterative-BP-CNN

channel noise. Because of the decoding error, the channel noise estimation is not exact. We then feed the channel noise estimation into a CNN, which further removes the estimation errors of the BP decoder and obtains a more accurate noise estimation by exploiting the noise correlation via training. Iterating between BP and CNN will gradually improve the decoding SNR and hence result in better decoding performance.

The proposed iterative BP-CNN decoder has many desirable properties. As we will see, it has better decoding performance than the standard BP, with lower complexity. This is mainly due to the efficient CNN structure, which consists mostly of linear operations and a few non-linear ones. It allows the system to learn the network parameters directly from data, without any knowledge of the channel model. The overall BP-CNN architecture can remain the same when the underlying use cases change, and we only need to re-train the network and update its parameters to adapt to the new use case. It is also suitable for parallel computing, which is important for VLSI implementation. Furthermore, our experiments show that the iterative BP-CNN decoder is robust to SNR and correlation mismatches if the training data is generated carefully.

Intuitively, the reason that CNN can help channel decoding is similar to the success of CNN in low-level tasks in image processing such as image denoising [25] or image super-resolution [26]. This becomes more clear when we view the correlation in channel noise as a "feature", which can be extracted by CNN. However, our problem setting is very different to these other applications where extracting features is the ultimate goal. In the iterative BP-CNN architecture, the goal of CNN is not only to accurately estimate the channel noise and depress the residual error, but also to generate an output that is benign to the BP decoder. This unique requirement has motivated us to develop a novel loss function for CNN training, which combines the effect of residual noise power with a Jarque-Bera normality test [27]. We will see that this new loss function plays an important role in the *enhanced BP-CNN* decoder.

In summary, our contributions in this work are as follows:

1) We propose a novel decoding architecture for linear codes, called *iterative BP-CNN*, that concatenates a trained CNN with a standard BP decoder and iterates between them. This architecture is shown to have the capability to extract noise correlation features and improve the decoding performance.
2) We design the CNN architecture for *iterative BP-CNN* with two different loss functions. In the *baseline BP-CNN*, the well known quadratic loss function is used in CNN training. In the *enhanced BP-CNN*, we develop a new loss function that depresses the residual noise power and simultaneously performs the Jarque-Bera normality test.
3) We carry out extensive evaluations to verify and analyze the proposed framework.

The rest of this paper is organized as follows. In Section II we give a brief review of some related works. The system design, including the network training, is explained in detail in Section III. Extensive experiments is in Section IV. We finally conclude this paper and discuss future works in Section V.

II. Related Works

A. Convolutional Neural Networks

Recent progresses of deep learning technologies, big data, and powerful computational devices such as GPUs have started a new era in artificial intelligence. Taking computer vision as an example, deep convolutional neural networks have been verified to yield much better performance than conventional methods in various applications, from high-level tasks such as image recognition [11], [28] and object detection [29], to low-level tasks such as image denoising [25] and image super-resolution [26]. In [28], the authors propose to use a deep CNN for image classification, consisting of convolutional layers, pooling layers, and fully-connected layers. A variant of this model won the ILSVRC-2012 competition, and CNNs have received skyrocketed interest in both academia and industry since then. In [11], the authors have designed a residual learning framework (ResNet) which makes it easier to train deeper neural networks. It is shown that ResNet can even outperform humans in some high-level tasks. With regard to low-level tasks such as image super-resolution, the authors of [26] show that a fully convolutional neural network can achieve similar restoration quality with the-state-of-art methods but with low complexity for practical applications. More recently, CNNs have played a crucial role in AlphaGo [30] which beats the best human player in Go games.

The task of channel decoding under colored noise is similar to some low-level tasks such as image denoising or super-resolution, which has motivated us to combine a CNN with BP in this work. We will elaborate on this similarity in Section III.

B. Applications of Deep Learning in Communications

Most algorithms in communications are developed based on given models. For some tasks, this is only feasible when the model is simple enough, and model mismatch often happens in practice. Recent advances in deep learning technologies have attracted attention of researchers in communications and there have been some recent papers on solving communication problems using deep learning technologies. For the topic of designing channel decoding algorithms, which is the focus of this paper, Nachmani *et al.* have demonstrated that by assigning proper weights to the passing messages in the Tanner graph, comparable decoding performance can be achieved with less iterations than traditional BP decoders [14]. These weights are obtained via training in deep learning, which partially compensate for the effect of small cycles in the Tanner graph. The authors further introduce the concept of recurrent neural networks (RNN) into BP decoding in [17]. Combined with the modified random redundant (mRRD) iterative algorithm [31], additional performance improvement can be obtained. Considering that BP decoding contains many multiplications, Lugosch *et al.* have proposed a light-weight neural offset min-sum decoding algorithm in [18], with no multiplication and friendly for hardware implementation. Addressing the challenge that deep learning based decoders are difficult to train for long codes [19], the authors in [20] propose to divide the polar coding graph into sub-blocks, and the decoder for each sub-codeword is trained separately.

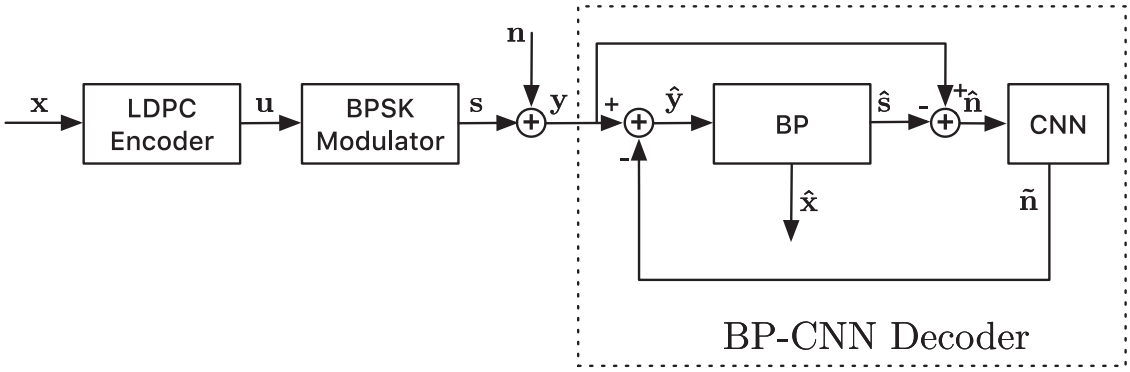


Fig. 1. The proposed iterative decoding architecture which consists of a belief propagation (BP) decoder and a feed-forward convolutional neural network (CNN).

Besides channel decoding, deep learning has the potential to achieve improvement in other areas of communication systems [21]. O'Shea *et al.* propose to learn a channel auto-encoder via deep learning technologies [15]. Farsad *et al.* study the problem of developing detection algorithms for communication systems without proper mathematical models, such as molecular communications [22]. The authors of [23] apply deep neural networks to MIMO detection, which results in comparable detection performance but with much lower complexity. In the application layer, deep recurrent neural networks can be also used to recognize different traffic types [16]. Most recently, Dorner *et al.* [24] have demonstrated the feasibility of over-the-air communication with deep neural networks and software-defined radios.

These early studies have not considered the channel decoding problem under correlated Gaussian noise, which is another example of complex channel models that are difficult for theoretical analysis and algorithm development. In this paper, we focus on this problem and propose a novel iterative BP-CNN receiver for channel decoding.

III. System Design

This section contains the main innovation of this work: an iterative BP-CNN channel decoder. In order to better present the proposed design, we will first describe the system framework. Specifically, the CNN structure will be introduced and its role in helping channel decoding will be explained. Finally, we will highlight two crucial components in training the network: the design of the loss function and the data generation.

A. System Framework

The proposed architecture is illustrated in Fig. 1. At the transmitter, a block of uniformly distributed bits $\mathbf{x}$ of length K, is encoded to a binary codeword $\mathbf{u}$ of length N through a linear channel encoder. In this paper, we focus on the LDPC code, but the proposed method is readily applicable to other linear block codes. The codeword $\mathbf{u}$ is then mapped to a symbol vector $\mathbf{s}$ through the BPSK modulation.

The BPSK symbols will be passed through a channel with additive Gaussian noise. The channel noise vector, denoted as $\mathbf{n}$ of length N, is modeled as a Gaussian random vector with auto-correlation matrix $\mathbf{\Sigma}$. Note that the LDPC codeword may be long, and as a result the size of $\mathbf{\Sigma}$ can be significant. In this paper, we do not require any specific format for $\mathbf{\Sigma}$. As we will verify later, the proposed architecture is effective under different formats of the correlation model.

At the receiver, vector $\mathbf{y}$ is received and can be written as

$$\mathbf{y} = \mathbf{s} + \mathbf{n}. \tag{1}$$

The signal to noise ratio (SNR) is defined as

$$\mathsf{SNR} = 10 \log \left(\frac{P}{\sigma^2} \right), \tag{2}$$

where P is the average transmit power $P = \mathbb{E}\left[s^2\right]$ and σ^2 in Eqn. (2) is defined as the power of the complex noise samples. Since the coding performance is evaluated with BPSK, the effective noise only comes from the I-samples. Therefore, the effective noise power is $\sigma^2/2$. Here we assume that the complex Gaussian noise is circularly symmetric.

A BP decoder is then used to decode the transmitted information bit vector from $\mathbf{y}$. Typically, there is a BPSK soft demodulator before the BP decoding, to calculate the log-likelihood ratios (LLRs) of the transmitted symbols with the knowledge that the channel noise follows a Gaussian distribution with power $\sigma^2/2$.

$$\mathsf{LLR}_i^{(1)} = \log \frac{\Pr(y_i|s_i = 1)}{\Pr(y_i|s_i = -1)} = \frac{2y_i}{\sigma^2/2}, \tag{3}$$

where s_i, y_i denote the ith BPSK symbol and the corresponding received symbol, respectively, and the superscript (1) to LLR indicates that this is the LLR computation for the initial BP. In our system framework of Fig. 1, we omit this module and merge it into the BP decoder for simplicity.

We use $\hat{\mathbf{s}}$ to denote the estimated transmit symbols. Subtracting it from the received symbols $\mathbf{y}$, we obtain $\hat{\mathbf{n}}$ as

$$\hat{\mathbf{n}} = \mathbf{y} - \hat{\mathbf{s}}, \tag{4}$$

which can be viewed as an estimation of the channel noise. Because of the decoding errors in the BP decoder, $\hat{\mathbf{n}}$ is not exactly the same as the true channel noise $\mathbf{n}$. We can re-write $\hat{\mathbf{n}}$ as

$$\hat{\mathbf{n}} = \mathbf{n} + \boldsymbol{\xi}, \tag{5}$$

where $\boldsymbol{\xi}$ is the error vector of noise estimation.

We are now ready to explain the proposed BP-CNN decoding architecture, as shown in the dashed box of Fig. 1. Inspired

by the successful application of CNNs in image denoising and super-resolution, and noticing that the correlation in channel noise $\mathbf{n}$ can be considered as a "feature" that may be exploited in channel decoding, we propose to concatenate a CNN after BP, to utilize this correlation to suppress $\boldsymbol{\xi}$ and get a more accurate estimation of the channel noise. The details of the proposed CNN and the associated training procedure will be presented in subsequent sections. Using $\tilde{\mathbf{n}}$ to denote the CNN output and subtracting it from the received vector $\mathbf{y}$ result in

$$\hat{\mathbf{y}} = \mathbf{y} - \tilde{\mathbf{n}} = \mathbf{s} + \mathbf{n} - \tilde{\mathbf{n}} = \mathbf{s} + \mathbf{r} \tag{6}$$

where $\mathbf{r} = \mathbf{n} - \tilde{\mathbf{n}}$ is defined as the *residual noise*. Then, the new vector $\hat{\mathbf{y}}$ is fed back to the BP decoder and another round of BP decoding is performed. Note that before the second round of BP iterations, the LLR values need to be updated as follows:

$$\mathsf{LLR}_i^{(2)} = \log \frac{\Pr(\hat{y}_i | s_i = 1)}{\Pr(\hat{y}_i | s_i = -1)}, \tag{7}$$

where $\hat{y}_i$ denotes the ith element of $\hat{\mathbf{y}}$, and the superscript (2) to LLR indicates that this is the LLR computation for the subsequent BPs after processing the noise estimate with CNN. The above process can be executed multiple times to gradually suppress the residual noise. The LLR computation in these following rounds of iterations will be similar to Eqn. (7) with the most recently processed vectors.

Obviously, the characteristics of the residual noise $\mathbf{r}$ will affect the calculation in (7) and thereby influence the performance of the subsequent BP decoding. Therefore, the CNN should be trained to provide residual noise that is beneficial for the BP decoding. We propose two strategies for training the CNN and concatenating with BP.

1) *Baseline BP-CNN.* The CNN is trained to output a noise estimation which is as close to the true channel noise as possible. This is a standard method which can be implemented using a quadratic loss function in CNN training. Intuitively, an accurate noise estimation will result in very low power residual noise and thus very little interference to the BP decoding. In order to calculate the LLR in (7), we also need to obtain an empirical probability distribution of the residual noise, denoted as $\hat{P}_{\boldsymbol{\xi}}(\cdot)$. This can be done after the network training is completed using the training data. Therefore, we have

$$\mathsf{LLR}_i^{(2)} = \log \frac{\Pr(\hat{y}_i | s_i = 1)}{\Pr(\hat{y}_i | s_i = -1)} = \log \frac{\hat{P}_{\xi}(\hat{y}_i - 1)}{\hat{P}_{\xi}(\hat{y}_i + 1)}. \tag{8}$$

2) *Enhanced BP-CNN.* The baseline BP-CNN may not be optimal considering that the channel encoding (such as LDPC) is mostly optimized for the AWGN channel and the residual noise may not necessarily follow a Gaussian distribution. Therefore, another strategy is to depress the residual noise power and simultaneously let the residual noise follow a Gaussian distribution as much as possible. By doing this, the LLR in (7) is calculated with the assumption that the residual noise follows a Gaussian distribution:

$$\mathsf{LLR}_i^{(2)} = \frac{2\hat{y}_i}{\sigma_{\mathbf{r}}^2}, \tag{9}$$

where $\sigma_{\mathbf{r}}^2$ is the power of the residual noise which can be estimated using the training data. Compared to the *baseline BP-CNN*, the *enhanced BP-CNN* has the following advantages. First, the calculation of LLRs becomes much easier. It needs to only estimate and store the power of the residual noise rather than the empirical probability distribution. Second, it may yield a better performance considering the channel encoding is mostly optimized for the AWGN channel. This will be verified in Section IV.

In Section III-E, we will show how to train the network with the above two strategies. With a depressed influence of noise, the BP decoding is expected to yield a more accurate result. The above processes can be executed iteratively to successively depress the noise influence and improve the final decoding performance.

B. Why is CNN Useful for Channel Decoding?

As we have mentioned, one of the motivations to use a CNN to estimate the channel noise comes from CNN's successful applications in some low-level tasks such as image denoising [25], [32] and image super-resolution [26]. A careful investigation into these CNN applications reveals important similarity to the channel decoding task at hand. Let us take image denoising as an example. Note that the task of image denoising is to recover the original image pixels $\mathbf{X}$ from its noisy observation $\mathbf{Y}$, following the additive model $\mathbf{Y} = \mathbf{X} + \mathbf{W}$ where $\mathbf{W}$ denotes the unknown noise. As analyzed in [25], there is a strong mathematical relationship between convolution networks and the state-of-the-art Markov random field (MRF) methods. One advantage to utilize a CNN for image denoising is that it allows the network to learn high dimensional model parameters to extract image features, which have stronger image processing power. The results in [25] demonstrate that using CNN can achieve a better performance with low complexity than previously reported methods.

Returning to our task in this paper, the role of CNN in the decoding architecture is to recover the true channel noise $\mathbf{n}$ from the noisy version $\hat{\mathbf{n}}$ following model (5), which is very similar to image denoising. The recovered channel noise $\mathbf{n}$ in our task corresponds to the image pixels $\mathbf{X}$ in image denoising and the error $\boldsymbol{\xi}$ corresponds to the noise $\mathbf{W}$. Note that just like CNN-based image denoising which exploits the image features in pixels $\mathbf{X}$, the iterative BP-CNN decoder exploits the correlations in channel noise $\mathbf{n}$. Given that CNN has excellent denoising performance with low complexity, it is natural to ask if such advantage can translate to the channel decoding task in this paper.

C. Belief Propagation Decoding

In order to make this paper self-contained, we briefly introduce the standard belief propagation (BP) decoding process. BP is an iterative process in which messages are passed between the variable nodes and check nodes in a Tanner graph. We use v to denote a variable node and c for a check node. $L_{v \to c}$ and $L_{c \to v}$ are messages passed from v to c and c to v, respectively. First, $L_{v \to c}$ is initialized with the LLR value calculated from the received symbol, as shown in (3). Then the messages are

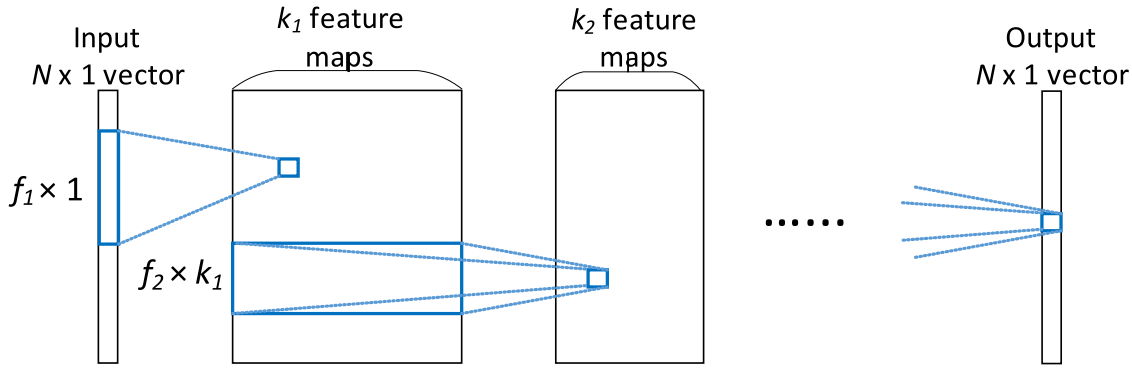


Fig. 2. The adopted CNN structure for noise estimation.

updated iteratively as follows

$$L_{v\to c} = \sum_{c'\in\mathcal{N}(v)\backslash c} L_{c'\to v},$$

$$L_{c\to v} = 2\tanh^{-1}\left(\prod_{v'\in\mathcal{N}(c)\backslash v}\tanh\frac{L_{v'\to c}}{2}\right), \quad (10)$$

where $\mathcal{N}(v)\backslash c$ ($\mathcal{N}(c)\backslash v$) denotes the set of neighboring check (variable) nodes of the variable node v (the check node c), except c (v). After several iterations, the LLR of a variable node v, denoted as L_v, is calculated as

$$L_v = \sum_{c'\in\mathcal{N}(v)} L_{c'\to v}. \quad (11)$$

Finally, the bit value corresponding to the variable node v is determined by

$$b = \begin{cases} 0, & \text{if } L_v \geq 0, \\ 1, & \text{if } L_v \leq 0. \end{cases} \quad (12)$$

The standard BP decoding is well known to be effective in AWGN channels [33] but will have difficulty to handle correlation in the channel noise. To see this point, we recall that the BP decoding algorithm operates on a factor graph with the following probability model for the AWGN channel:

$$p(u_1, u_2, \ldots, u_N) = \left(\prod_i p_{ch}(u_i)\right)\left(\prod_i f_i(\mathcal{N}_i)\right), \quad (13)$$

where u_i is the ith coded bit, $p(u_1, u_2, \ldots, u_N)$ denotes the joint posterior probability of all coded bits, which is determined by the channel observations and the coding structure. $p_{ch}(u_i)$ is the probability conditioned on the received channel symbols, $f_i(\cdot)$ denotes the ith parity-check indicator function corresponding to the ith check node, and $\mathcal{N}_i$ denotes the set of variable nodes connected to the ith check node. For a decoder facing correlated noise, however, the probability model can only be presented in the following form:

$$p(u_1, u_2, \ldots, u_N) = p_{ch}(u_1, u_2, \ldots, u_N)\left(\prod_i f_i(\mathcal{N}_i)\right), \quad (14)$$

where the probability conditioned on the received symbols cannot be factored as symbols are correlated. Therefore, improving the standard BP decoding algorithm to incorporate the joint distribution of noise samples may have very high complexity, especially when the channel noise has strong correlations.

D. CNN for Noise Estimation

As mentioned before, adopting CNN for noise estimation is enlightened by its successful applications in computer vision, which demonstrate its strong capability to extract local features. For some specific image restoration tasks, Dong *et al.* [26] have shown that CNN has a similar performance as previously known strategies which represent image patches by a set of pre-trained bases, but with a lower complexity. Moreover, the network can be trained to find better bases instead of the pre-defined ones, for better performance. Therefore, CNN has the potential to yield a better image restoration quality.

In the proposed iterative BP-CNN architecture, the adopted network structure, which is shown in Fig. 2, is similar to those used for low-level tasks in image restoration [25], [26] but with a conspicuous difference in that the input of our network is a 1-D vector instead of a 2-D image. As can be seen at the first layer in Fig. 2, k_1 feature maps are generated from the input data $\hat{\mathbf{n}}$, which can be expressed as

$$\mathbf{c}_{1,j} = \mathsf{ReLU}(\mathbf{h}_{1,j} * \hat{\mathbf{n}} + b_{1,j}), \quad (15)$$

where $\mathbf{c}_{1,j}$ is the jth feature map at the first layer, $\mathbf{h}_{1,j}$ is the jth convolution kernel, which is essentially a 1-D vector of length f_j, $b_{1,j}$ is the corresponding scalar bias, and $\mathsf{ReLU}(\cdot)$ is the Rectified Linear Unit function ($\max(x, 0)$) in order to introduce nonlinearity [34]. In (15), the addition means that the scalar bias is added to *all* the output elements of the convolution, which is a concise and commonly used representation in deep learning [25], [26]. In neural networks, the convolution operation is denoted by $*$ in (15) and is defined as

$$(\mathbf{h}_{1,j} * \hat{\mathbf{n}})(v) = \sum_{\Delta v} \hat{\mathbf{n}}(v + \Delta v)\mathbf{h}_{1,j}(\Delta v), \quad (16)$$

which is also slightly different with the standard definition in signal processing.

At the ith layer ($i > 1$), the convolution operation is performed over all feature maps of the previous layer, which can

be viewed as a 2-D convolution as shown in Fig. 2. The output can be written as

$$\mathbf{c}_{i,j} = \mathsf{ReLU}(\mathbf{h}_{i,j} * \mathbf{c}_{i-1} + b_{i,j}), \tag{17}$$

where $\mathbf{c}_{i,j}$ is the jth feature map at the ith layer. $\mathbf{h}_{i,j}$ is the jth convolution kernel of size $f_i \times k_{i-1}$ where k_{i-1} is the number of feature maps of the previous layer. We use L to denote the number of total layers. At the last layer, the final estimation of the channel noise is

$$\tilde{\mathbf{n}} = \mathbf{h}_L * \mathbf{c}_{L-1} + b_L. \tag{18}$$

To summarize, the structure of a CNN is determined by its number of layers, filter sizes, and feature map numbers in each layer. These parameters need to be decided before training the network. For simplicity of exposition, we denote the structure of the network as

$$\{L; f_1, f_2, .., f_L; k_1, k_2, \ldots, k_L\}. \tag{19}$$

Besides convolution layers, *pooling*, *dropout* and *fully-connected* layers are also important components in ordinary CNN architectures. However, they are not considered in our design for the following reasons. First, pooling layers are often used for downsampling in high-level tasks. In our problem, the CNN output is still a low-level representation which has the same dimension as the CNN input. Pooling layers are not needed in this case, because it may discard some useful details [35]. Second, dropout is a widely used technique to control overfitting. In this work, we already have overfitting controlled by tracking the loss value in the validation set. Furthermore, we have tested the performance of dropout and found that it does not provide any gain. With regard to the fully connected layers, they are often used to output the high-level description after the data dimensions have been reduced to a large extent. In the low-level tasks, fully-connected layers are of high complexity and also hard to train.

E. Training

1) Loss Function: It is well known in deep learning that the performance of a network depends critically on the choice of loss functions for training [36]. Generally speaking, a loss function is to measure the difference between the actual CNN output and its expected output, and it should be carefully defined based on the specific task of the network. In the proposed architecture, CNN is used to estimate the channel noise and its output will influence the performance of BP decoding in the next iteration. Therefore, a proper loss function must be selected by fully considering the relationship between CNN and the following BP decoding.

As introduced in Section III-A, to facilitate the subsequent BP decoding, there are two strategies to train the network. One is to only depress the residual noise power (*baseline BP-CNN*), and the other one is to depress the residual noise power and simultaneously shape its distribution (*enhanced BP-CNN*).

For *baseline BP-CNN*, the loss function can be chosen as the typical residual noise power:

$$\mathsf{Loss}_{\mathrm{A}} = \frac{\|\mathbf{r}\|^2}{N}, \tag{20}$$

where N is the length of a coding block. Note that this is the well adopted quadratic cost function in neural network training.

For *enhanced BP-CNN*, how to define a proper loss function is an open problem. Well-known loss functions such as quadratic, cross-entropy, or Kullback-Leibler divergence do not accomplish the goal. In this work, we introduce *normality test* to the loss function, so that we can measure how likely the residual noise samples follow a Gaussian distribution. The new loss function is formally defined as

$$\mathsf{Loss}_{\mathrm{B}} = \frac{\|\mathbf{r}\|^2}{N} + \lambda \left(S^2 + \frac{1}{4}(C-3)^2 \right). \tag{21}$$

The first term in (21) measures the power of the residual noise and the second term, adopted from the Jarque-Bera test [27], represents a normality test to determine how much a data set is modeled by a Gaussian distribution. λ is a scaling factor that balances these two objectives. Specifically, S and C in (21) are defined as follows:

$$S = \frac{\frac{1}{N}\sum_{i=1}^{N}(r_i - \bar{r})^3}{\left(\frac{1}{N}\sum_{i=1}^{N}(r_i - \bar{r})^2\right)^{3/2}},$$
$$C = \frac{\frac{1}{N}\sum_{i=1}^{N}(r_i - \bar{r})^4}{\left(\frac{1}{N}\sum_{i=1}^{N}(r_i - \bar{r})^2\right)^{2}}, \tag{22}$$

where r_i denotes the ith element in the residual noise vector, and $\bar{r}$ is the sample mean. In statistics, S and C are called *skewness* and *kurtosis*, respectively. Although the Jarque-Bera test is not the optimal normality test, the cost function is derivable and simple for training. Furthermore, experimental results show that it can provide a very desirable output.

2) Generating the Training Data: To train the network, we need both channel noise data $\mathbf{n}$ and estimated noise data $\hat{\mathbf{n}}$ from the BP decoding results. For the simulation results in Section IV, we focus on the channel models with known (to the receiver) noise correlation functions, and thus we are able to generate adequate channel noise samples to train the network. For example, given the channel correlation matrix $\mathbf{\Sigma}$ in (24), the channel noise samples can be generated by

$$\mathbf{n} = \mathbf{\Sigma}^{1/2}\mathbf{n}_w, \tag{23}$$

where $\mathbf{n}_w$ is a vector of independent identically distributed, standard Gaussian random variables. The estimated noise $\hat{\mathbf{n}}$ can be obtained by generating uniform distributed binary bits $\mathbf{x}$ and successively performing channel encoding, BPSK mapping, simulated channel interference and BP decoding.

Another factor in generating the training data is the channel condition, i.e., SNR, which will determine the severity of errors of the BP decoding and hence affect the input to the network. We use Γ to denote the set of SNRs used to generate training data. If the channel condition is very good, very few errors exist $\hat{\mathbf{n}}$ and the network may not learn the robust features of the channel noise. On the other hand, if the channel condition is very bad, many errors exist in $\hat{\mathbf{n}}$ and they will mask the channel noise features, which is also detrimental for the network training. In Section IV, we will conduct some experiments to analyze this problem.

We want to emphasize that although CNN training typically requires a large amount of data, has high complexity, and the resulting network depends on the training data, these factors will not impede the application of the proposed architecture to practical systems. The network training is largely done *offline*, where the required training complexity can be satisfied with powerful computational devices such as GPUs. The amount of training data is also an offline issue, which can be generated and stored in mass storage. Lastly, although CNN depends on the training data, in the practical communication system design we often focus on certain representative scenarios ("use cases") and we can generate the training data to reflect these scenarios. The resulting trained CNNs can be stored in the on-device memory for online use. As an alternative, the training data can also be collected from real-world deployment where no pre-defined channel model is assumed. However, although we believe that our method works with real-world data, we currently do not have the capability to carry out this experiment and will re-visit it in a future work.

F. Design Summary

The proposed design can be summarized using the relevant parameters, which will result in the tradeoff between performance and complexity as we will see later in the simulations. First, noting that the proposed decoding framework is an iterative architecture, one important parameter is the number of total iterations, denoted as K. Then, the structure of CNN is decided by the number of layers, filter sizes and the number of feature maps in each layer, denoted as $\{L; f_1, f_2, \ldots, f_L; k_1, k_2, \ldots, k_L\}$. For the BP decoder, the parameter of interest is the number of iterations. Essentially, the iterative BP-CNN decoding architecture in Fig. 1 can be unfolded to generate an open-loop structure and all networks are trained sequentially, which can be concisely denoted as BP-CNN1-BP-CNN2-...-CNNx-BP. The proposed iterative architecture is actually a subset of this general framework. Note that although the open-loop framework is more general and thus may result in better performance, it will consume a much larger amount of resource to implement. The closed-loop architecture in Fig. 1 only requires training and storing one CNN, while the open-loop framework has x CNNs. The training of multiple serially-concatenated CNNs may be highly complex, as a later CNN may depend on the training of an early one. Storage of x CNNs on the device will also increase the cost.

To train a network with *enhanced BP-CNN*, we need to select a proper λ to balance the significance of normality test in the loss function and the channel conditions to generate training data. Important system parameters that need to be carefully chosen are summarized in Table I. In Section IV, we will carry out extensive experiments to analyze the influence of these system parameters on the performance and provide some guidance on their selection.

TABLE I
SUMMARY OF IMPORTANT SYSTEM PARAMETERS

Notation	Meaning
K	The number of iterations between the BP decoder and the CNN
$\{L; f_1, \ldots, f_L; k_1, \ldots, k_L\}$	CNN structure: numbers of layers, filter sizes and numbers of feature maps
$\{B\}$	Numbers of BP iterations
λ	The scaling factor of the normality test for *enhanced BP-CNN*
Γ	The set of signal-to-noise ratios for generating training data
SNR	The ratio of signal power to the complex noise power in the I-Q plane

TABLE II
BASIC CNN SETTING FOR EXPERIMENTS

CNN structure	$\{4; 9, 3, 3, 15; 64, 32, 16, 1\}$
Weight number	8496
Edge number	4893696
Mini-batch size	1400
Size of the training data	2000000
Size of the validation data	100000
Channel SNRs for generating the training data (Γ)	$\{0, 0.5, 1, 1.5, 2, 2.5, 3\}$ dB
Initialization method	Xavier initialization
Optimization method	Adam optimization

IV. EXPERIMENTS

A. Overview

Throughout all experiments, we use a systematic LDPC code of rate $3/4$. The code block length is 576 and the parity check matrix is from [37]. We use random codewords rather than the all-zero codeword for simulation. The experimental platform is implemented in *TensorFlow* [38]. The basic CNN structure is $\{4; 9, 3, 3, 15; 64, 32, 16, 1\}$. The network is initialized using the Xavier initialization method, which has been demonstrated to perform better than random initialization [39].

Before training the network, the training data is first generated. Following the common practice in machine learning, some validation data is also generated to test the network loss and avoid potential overfitting. The training data is generated under multiple SNRs: $\{0, 0.5, 1, 1.5, 2, 2.5, 3\}$ dB. The data of each SNR occupies the same proportion.

We adopt the conventional mini-batch gradient descent method to train the network. Each mini-batch contains 1400 blocks of data and the data of each SNR occupies the same proportion in one mini-batch. In each iteration, a fixed number of training samples, i.e. a mini-batch, are randomly selected to calculate the gradient. We use the Adam optimization method for searching the optimal network parameters [40]. During the training process, we check the loss value over the validation set every 500 iterations. Training continues until the loss does not drop for a consecutive period of time (eight checks in our experiment, which equals to 4000 iterations). The basic CNN setting is summarized in Table II.

In all experiments, the setting is the same as Table II unless otherwise specified. The system performance is measured by the bit error rate (BER). In order to test BERs with high accuracy and low complexity, a total of 10^8 information bits are tested for high SNRs and 10^7 bits for low SNRs. We note that the

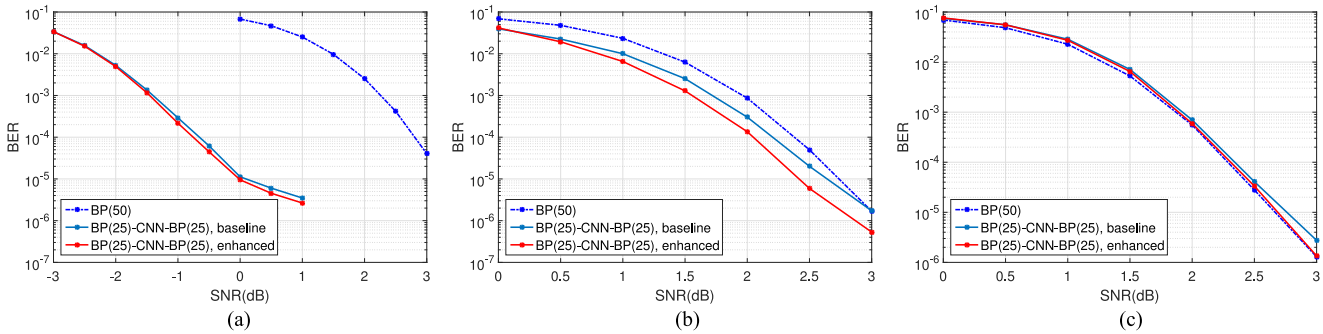


Fig. 3. Performance comparison of BP-CNN with the standard BP decoding. Only one iteration between CNN and BP decoder is executed. The numbers in the brackets denote the BP iterations. The SNRs corresponding to the Shannon capacities for $\eta = 0.8, 0.5$, and 0 are -6.9 dB, -2.4 dB and -0.4 dB, respectively. (a) $\eta = 0.8$, strong correlation. $\lambda = 0.1$. (b) $\eta = 0.5$, moderate correlation. $\lambda = 10$. (c) $\eta = 0$, no correlation. $\lambda = 10$.

validation SNRs can be different from the training SNRs. In all evaluations, we have focused on the performance comparison when BER is 10^{-4}; hence the validation SNRs are selected to cover this BER value and its proximity.

To better illustrate our design, we will use a standard correlation model for simulation, which is widely adopted in the literature [5]. The correlation matrix $\mathbf{\Sigma}$ is given by:

$$\Sigma_{i,j} = \begin{cases} \eta^{j-i}, & i \leq j \\ (\eta^{i-j})^*, & i \geq j, \end{cases} \tag{24}$$

where $\Sigma_{i,j}$ is the (i,j)th element of $\mathbf{\Sigma}$, and η is the correlation coefficient with $|\eta| \leq 1$. We should emphasize that (24) is not fundamental to our proposed design, as the BP-CNN architecture does not require a specific format of the channel model. We will present simulation results to verify this statement.

In the remainder of this section, we will analyze and verify the proposed iterative BP-CNN decoding architecture from different perspectives.

B. Performance Evaluation

1) BP-CNN Reduces Decoding BER: We first compare the performances of the proposed method and the standard BP decoder. For the BP decoder, total 50 iterations are executed (denoted as "BP(50)" in the figure). For the proposed method, we use its simplest form for testing, i.e. there is only one iteration between the BP decoder and the CNN. In this case, the receiver structure can be simply denoted as BP-CNN-BP. We present the comparison results for two correlation parameters: $\eta = 0.8$ represents a relatively strong correlation model and $\eta = 0.5$ represents a moderate one. In addition, the test results under an AWGN channel without any correlation ($\eta = 0$) are presented to demonstrate that the proposed method can provide similar performance as the conventional BP decoder when no correlation exists. This suggests that the proposed method can support a wide range of correlation levels. Both *baseline* and *enhanced* BP-CNNs are tested. For *enhanced BP-CNN*, we set λ to 0.1, 10 and 10 for $\eta = 0.8$, 0.5 and 0, respectively.

In order to isolate and identify the contribution of CNN, we have kept the total number of BP iterations the same in two systems. In the BP-CNN method, we execute 25 BP iterations in each BP decoding process (denoted as "BP(25)-CNN-BP(25)" in the figure), resulting in the same 50 BP iterations as the standard BP decoder.

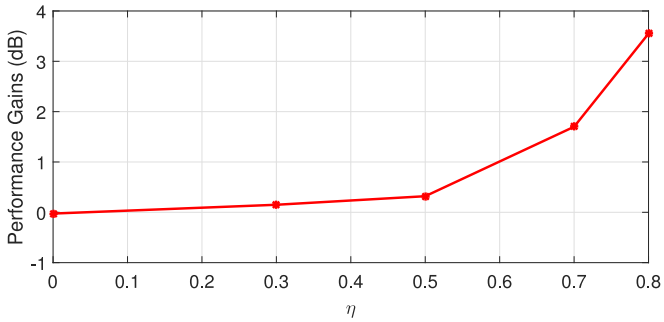


Fig. 4. Performance gains of *enhanced BP-CNN* under different η's. $\lambda = 10$ for $\eta = 0, 0.3, 0.5$ and $\lambda = 0.1$ for $\eta = 0.7, 0.8$.

The experiment results are reported in Fig. 3. We can see that both baseline and enhanced BP-CNN achieve significant performance gains with correlated noise. In the strong correlation case when $\eta = 0.8$, BP-CNN can improve the decoding performance by approximately 3.5 dB at BER $= 10^{-4}$. It should be emphasized that this performance gain cannot be compensated by more iterations in the standard BP decoder, as BP(50) already achieves a saturating performance. In the moderate correlation case with $\eta = 0.5$, the performance gain becomes smaller, because the correlation is weaker and the benefit of adopting a CNN is less. For the special case where $\eta = 0$ and the noise becomes i.i.d., which is the nominal AWGN channel, the proposed method performs similarly with the standard BP decoding. We thus conclude that the iterative BP-CNN decoding method can support a wide range of correlation levels, and the performance gains vary adaptively with the noise correlation. To see this more clearly, we plot the SNR gains at BER $= 10^{-4}$ of the enhanced BP-CNN decoder over the standard BP decoding algorithm under different values of η in Fig. 4. We see that the performance gain grows *monotonically* with the correlation level η, which is as expected because higher η presents a better opportunity for CNN to extract the noise "feature".

We can also compare the baseline and enhanced BP-CNN decoders from Fig. 3. We see that although both methods

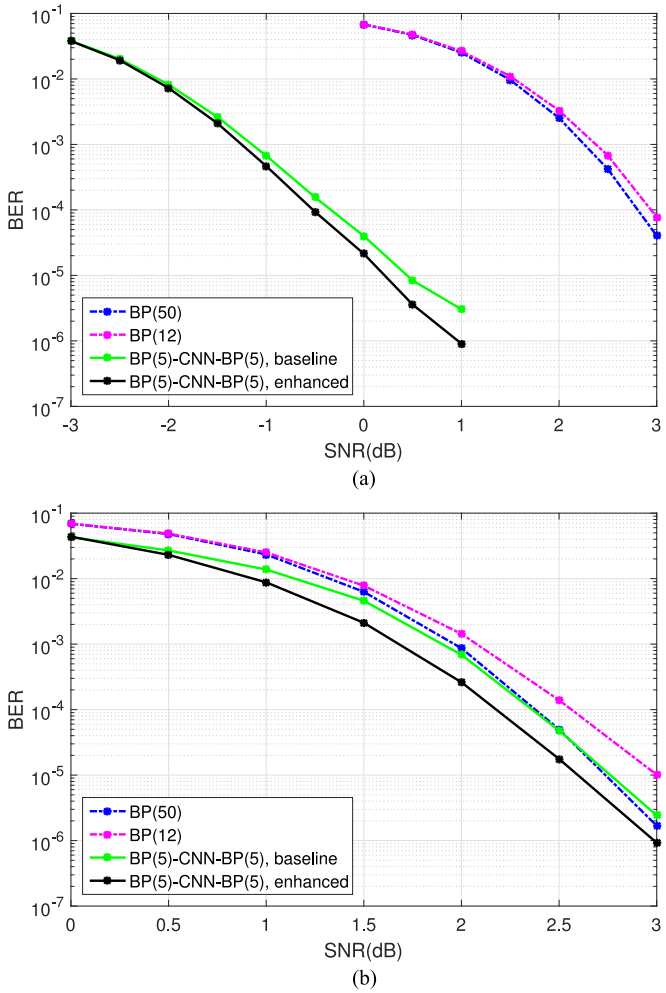


Fig. 5. BP-CNN achieves performance gains with lower complexity. (a) $\eta = 0.8$, strong correlation. $\lambda = 0.1$. (b) $\eta = 0.5$, moderate correlation. $\lambda = 10$.

outperform the standard BP, *enhanced BP-CNN* further outperforms the baseline strategy. As explained in Section III-E, the enhanced strategy balances both accurate noise estimation and shaping the output empirical distribution, and thus is better suited for concatenation with the BP decoder.

2) BP-CNN Achieves Performance Gain With Lower Complexity: Besides improving the decoding performance under the same number of BP iterations, another desirable feature of the iterative BP-CNN decoder is that it can outperform the standard BP decoding with lower overall complexity. To see this, we present another set of results in Fig. 5. The implementation details are the same as Fig. 3 except that in the BP-CNN decoder, the number of each BP decreases from 25 to 5. For comparison, we also plot the standard BP decoding performance with 12 and 50 iterations, respectively.

Ideally, we would like to set the BP parameters so that both methods have exactly the same overall complexity, and then compare their BER performances. However, an accurate comparison of the complexity associated with a BP network and a CNN is quite difficult. The main reason is that BP and CNN rely on very different types of operations. In CNN, there are only additions, multiplications and ReLU operations, while a BP network utilizes complex nonlinear operations such as $\tanh$, $\mathrm{arctanh}$, $\exp$ and $\log$. A comprehensive complexity comparison is an interesting research topic. We get around this problem by comparing their runtime under the same computation environment[1] and adjusting the parameters accordingly. We note that simply comparing their running time is not sufficiently accurate for the complexity comparison, as they largely depend on the implementation architectures. Nevertheless, this approach gives us an approximate comparison of their complexity and may serve as a baseline for future studies. Furthermore, the execution of CNN is highly optimized and well supported in *TensorFlow* and other neural network libraries, which is an important advantage of the proposed BP-CNN architecture. With the rapid development of AI chips, we envision that complexity will not be a major issue for the application of CNN in future communication systems.

In our test environment, we observe that the runtime of CNN with the structure $\{4; 9, 3, 3, 15; 64, 32, 16, 1\}$ is roughly equivalent to two BP iterations. This means that the chosen BP(5)-CNN-BP(5) structure in Fig. 5 has approximately the same complexity as the standard BP decoder with 12 iterations. We also plot BP(50) and observe that further increasing the complexity of standard BP provides some marginal gain. Hence, BP(12) represents a good-but-not-saturated scenario where the comparison to BP(5)-CNN-BP(5) is meaningful.

We see from Fig. 5 that the baseline BP(5)-CNN-BP(5) decoder has comparable performance of standard BP(50), but with a much lower complexity. When comparing the decoding performance with approximately the same complexity, both baseline and enhanced BP(5)-CNN-BP(5) decoders outperform the standard BP(12) decoder. Again, this gain is significant (around 3 dB) in the strong correlation case of Fig. 5(a) but still noticeable (0.1 to 0.5 dB) in the moderate correlation case of Fig. 5(b).

As the enhanced BP-CNN decoder outperforms the baseline one, we focus on the enhanced BP-CNN in the remainder of the experiments.

3) The Choice of Hyperparameter λ Affects the Performance of Enhanced BP-CNN: In (21), we have defined a new loss function for enhanced BP-CNN, which involves measures of both *power* and *normality* of the residual noise. The coefficient λ is an important hyperparameter to balance the trade-off between the residual noise power and its distribution. To demonstrate the importance of λ, we report the simulation results in Fig. 6, using different values of λ in the enhanced BP-CNN decoder. Note that the simulation uses the basic setting except the value of λ. To speed up the simulations, we use a low-complexity architecture of BP(5)-CNN-BP(5) in this set of experiments. We can see from Fig. 6 that very small values of λ cannot guarantee a Gaussian distribution for the residual noise, while very large λ cannot depress the residual noise power. Hence a proper choice of hyperparameter λ will affect the performance of BP-CNN. However, just like in many other deep learning tasks, optimizing

[1]Both are implemented in *TensorFlow* and simulations are run with the same computation resources.

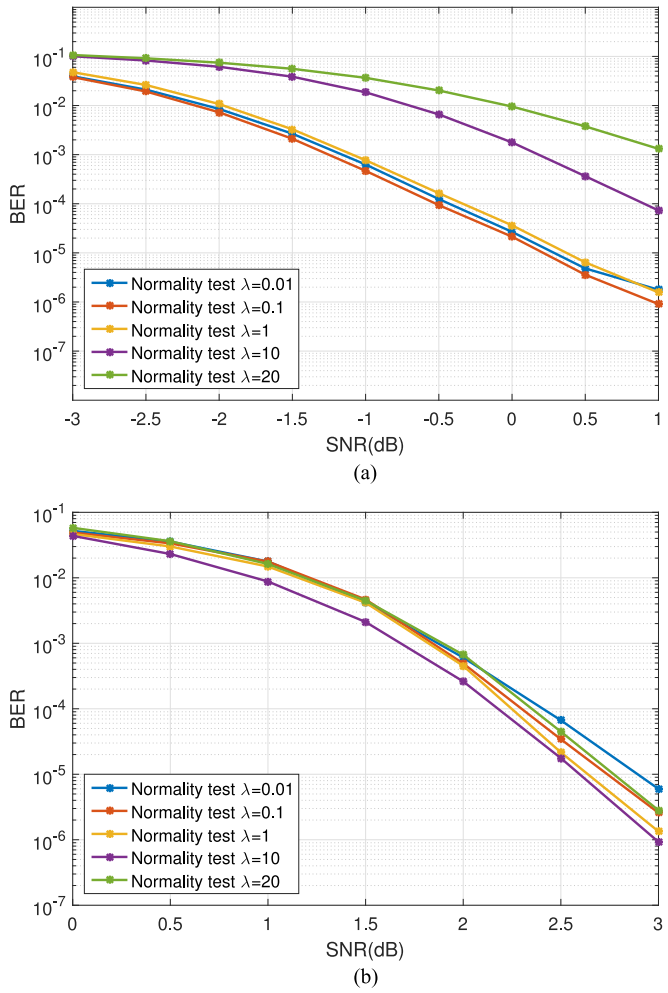


Fig. 6. Showing the necessity of the normality test. The receiver structure is BP(5)-CNN-BP(5). (a) $\eta = 0.8$, strong correlation. (b) $\eta = 0.5$, moderate correlation.

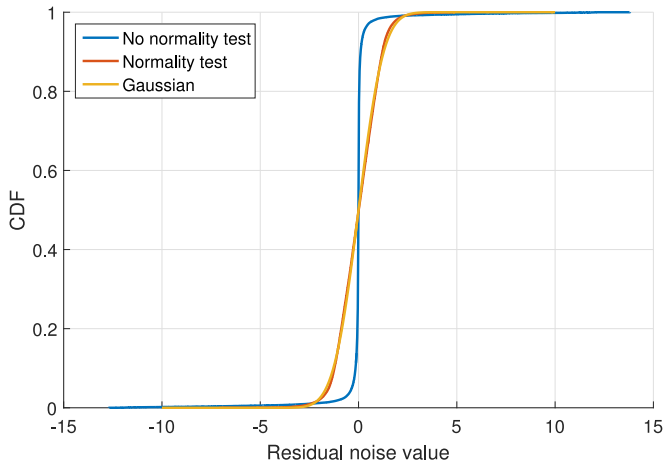


Fig. 7. Compare the residual noise distribution with and without normality test. The residual noise is collected under $\eta = 0.8$, SNR $= 0$ dB and the system structure is BP(5)-CNN-BP(5). When training the network with the normality test, λ is set to be 0.1. The distributions are normalized to have unit variance in order to compare with the Gaussian distribution.

λ analytically is very difficult and we select this parameter based on simulations.

Now let us take a deeper look at the new loss function (21), and numerically verify whether the normality test can indeed shape the output distribution to approximate Gaussian. We report the empirical cumulative distribution function (CDF) of the residual noise after CNN in Fig. 7, both with (i.e., *enhanced* BP-CNN) and without (i.e., *baseline* BP-CNN) normality test. The residual noise data is collected under $\eta = 0.8$, SNR $= 0$ dB and BP(5)-CNN-BP(5). When training the network with the normality test, λ is set as 0.1. It is evident from Fig. 7 that involving the normality test in the loss function makes the residual noise distribution more like Gaussian.

From Fig. 6, we also find that for $\eta = 0.8$, $\lambda = 0.1$ performs the best among all tested λ values, while $\lambda = 10$ performs the best for $\eta = 0.5$. We observe that a smaller λ is preferred when $\eta = 0.8$ compared to $\eta = 0.5$, which can be explained as follows. With strong correlation in the channel noise, more information can be utilized to remove the errors of the network input and thus a smaller λ is preferred to make the network focus on reducing the residual noise power. On the other hand, when correlation becomes weak and input elements are more independent, less information can be utilized to remove errors. In this case, the network needs to pay more attention to the distribution of the residual noise and a larger λ is thus favored.

4) Multiple Iterations Between CNN and BP Further Improve the Performance: Up to this point, we have only presented simulation results of the proposed iterative BP-CNN decoder with one iteration. Naturally, we can perform multiple iterations between CNN and BP with the hope of further reducing the BER, as shown in Fig. 1. Notation-wise, we use $K\{\text{BP}(n)\text{-CNN}\}$-BP$(n)$ to denote the iterative BP-CNN decoder structure with K iterations between BP and CNN, and n iterations inside BP. In total, $K+1$ BP(n) and K CNN are run. Enhanced BP-CNN is adopted for this set of experiments. We report the simulation results with different K's in Fig. 8. To show the performance improvement from multiple iterations, we also plot the performance of BP(25)-CNN-BP(25) for comparison. When $\eta = 0.8$, compared to the receiver architecture BP(5)-CNN-BP(5), we see that only increasing the BP iterations has limited performance gain. However, It is clear that multiple iterations can achieve larger improvement. Two BP-CNN iterations can improves the decoding performance by 0.7 dB at BER $= 10^{-4}$ compared to BP(5)-CNN-BP(5). In addition, we notice that after four BP-CNN iterations, the performance improvement becomes insignificant. This is because the CNN has reached its maximum capacity and cannot further depress the residual noise power.

5) BP-CNN is Robust Under Different Correlation Models: So far all the simulation results are obtained using the noise correlation model defined in (24). As mentioned in Sections III-A and IV-A, the proposed iterative BP-CNN decoder is not limited to a specific format of the correlation model. This should be intuitively reasonable as neither BP nor CNN knows the specific format of (24) in the previous experiments. When the format of

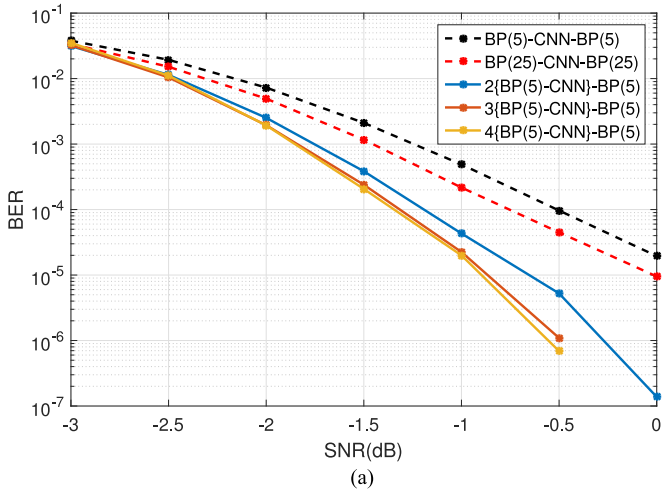


Fig. 8. Multiple iterations between CNN and BP can further improve decoding performance. (a) $\eta = 0.8$, strong correlation. (b) $\eta = 0.5$, moderate correlation.

the correlation model changes, the proposed method still works by re-training the network. We now verify this statement by simulating the iterative BP-CNN decoder with another correlation model whose characteristics are described by the following power spectrum density:

$$P(f) \propto 1/|f|^{\alpha}. \tag{25}$$

When $\alpha = 1$, the noise is commonly referred to as *pink noise*.

We simulate the enhanced BP-CNN with pink noise and the results are reported in Fig. 9. λ is set to 0.1 for training the network. Clearly, the proposed method still achieves significant performance gain, proving that the proposed method can support different models.

C. BP vs. CNN: Impact on the Decoding Performance

In the proposed framework, both the BP decoder and the CNN contribute to the improved decoding performance, but in very different ways. The BP decoder estimates the transmitted bits based on the encoding structure, while the CNN depresses the noise by utilizing its correlation. An important question arises:

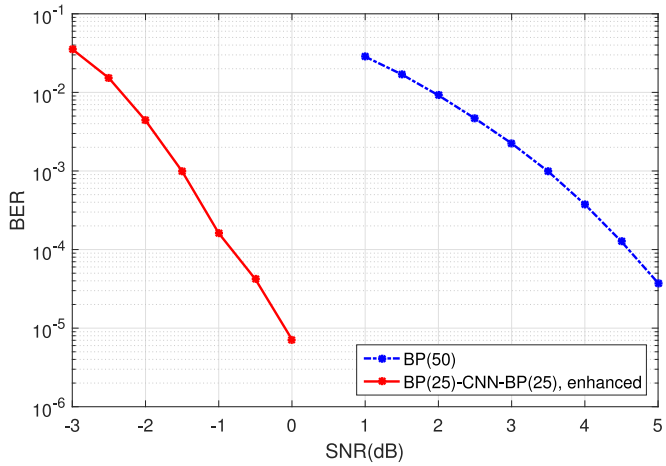
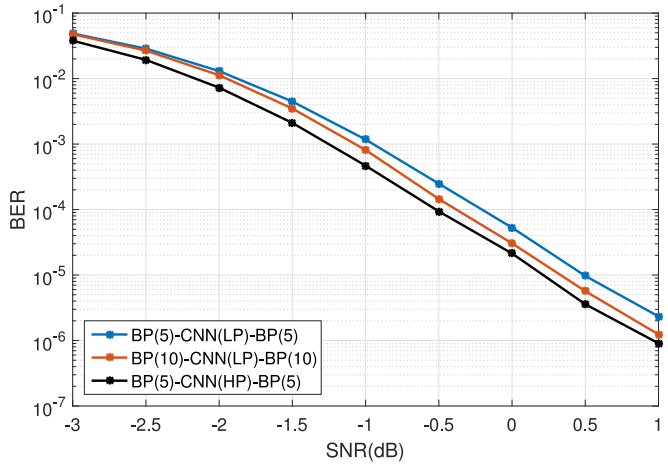


Fig. 9. Performance test with pink noise. $\lambda = 0.1$. The SNR corresponding to Shannon capacity is -5.5 dB.

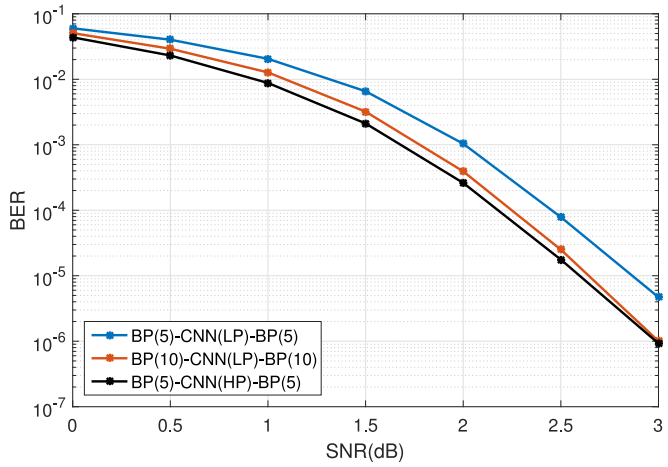
for a given amount of computation resources, how should the designer allocate the system complexity between BP and CNN, to obtain the best decoding performance?

A complete answer to this question remains unknown, and we resort to numerical simulations to gain some insight. In the simulations, we alter the complexity assignment between BP and CNN, and track the performance changes. We focus on the simplest form with only one iteration between BP and CNN, so that the receiver structure is concisely denoted as BP-CNN-BP. We start our investigation from a structure with relatively low complexity, and add complexity to the BP decoder and the CNN respectively to observe how the performance improves.

The results for $\eta = 0.8$ and $\eta = 0.5$ are given in Fig. 10. In each case, we first test a relatively low-complexity structure, denoted as BP(5)-CNN(LP)-BP(5), with five BP iterations executed in each BP decoding process. "CNN(LP)" denotes a low-complexity CNN, with its structure defined as $\{3; 5, 1, 9; 16, 8, 1\}$. Next, we add another five BP iterations to each BP decoding process to test a relatively high-complexity (with respect to BP) structure BP(10)-CNN(LP)-BP(10). Finally, we adopt a CNN structure $\{4; 9, 3, 3, 15; 64, 32, 16, 1\}$ (denoted as "CNN(HP)"), and test a relatively high-complexity (with respect to CNN) structure BP(5)-CNN(HP)-BP(5). First, it is clear that increasing the complexity of the BP decoder can improve the system performance, when the underlying CNN structure is unchanged. Second, as mentioned before, the complexity of the CNN(HP) structure is roughly equivalent to two BP iterations. Thus the receiver structure of BP(5)-CNN(HP)-BP(5) has lower complexity than BP(10)-CNN(LP)-BP(10). However, as observed from Fig. 10, BP(5)-CNN(HP)-BP(5) yields a better performance than BP(10)-CNN(LP)-BP(10). This demonstrates the necessity of introducing a CNN for decoding when the channel noise has strong or moderate correlation. The BP algorithm does not have the capability to extract the features existed in the channel noise for decoding. Therefore, when strong correlation exists, it is more effective to increase the CNN complexity to learn the channel characteristics than to increase the complexity of the BP decoder.



(a)



(b)

Fig. 10. The influences of different modules to the system performance. The legend CNN(LP) denotes a low-complexity network structure $\{3; 5, 1, 9; 16, 8, 1\}$ and CNN(HP) denotes a high-complexity network structure $\{4; 9, 3, 3, 15; 64, 32, 16, 1\}$. (a) $\eta = 0.8$, strong correlation. (b) $\eta = 0.5$, moderate correlation.

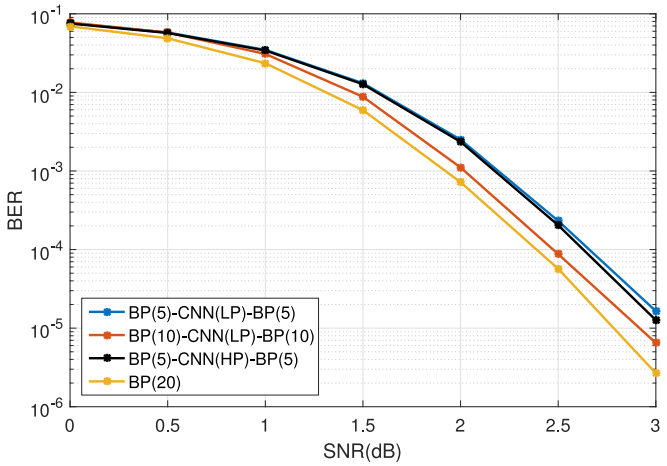


Fig. 11. Experiments under AWGN channels (i.e., $\eta = 0$) to show how the BP decoder and the CNN affect the system performance. The legend CNN(LP) denotes a low-complexity network structure $\{3; 5, 1, 9; 16, 8, 1\}$ and CNN(HP) denotes a high-complexity network structure $\{4; 9, 3, 3, 15; 64, 32, 16, 1\}$.

TABLE III
SETTINGS FOR TESTING THE IMPACT OF THE SELECTION OF Γ

	$\eta = 0.8$	$\eta = 0.5$
Receiver architecture	$\{4; 9, 3, 3, 15; 64, 32, 16, 1\}$	
CNN structure	BP(5)-CNN-BP(5)	
Validation SNR range	$-3 \sim 0$ dB	$0 \sim 3$ dB
Tested Γs	$\{-3, -2.5, -2, -1.5, -1, -0.5, 0\}$ dB	$\{0, 0.5, 1, 1.5, 2, 2.5, 3\}$ dB
	$\{-3\}$ dB	$\{0\}$ dB
	$\{0\}$ dB	$\{3\}$ dB

However, the above statement is not valid when the correlation is weak enough so that very few characteristics of the channel noise can be extracted by the CNN. To verify this, we report a set of tests under the AWGN channel in Fig. 11. Intuitively, in the AWGN case where channel has no "feature" for the CNN to extract, assigning more complexity to the BP decoder is more effective. This is verified in Fig. 11. We also add BP(20) as a baseline comparison. For the AWGN case, we see that BP(10)-CNN(LP)-BP(10) yields slightly worse performance than the standard BP(20) decoder. This mainly comes from the fact that CNN typically results in locally optimal solutions [41], and that concatenating two BP(10) decoders is worse than one BP(20) decoder.

D. Robustness to Mismatched SNRs

In the basic setting of our experiments, the training data is generated under multiple channel SNRs, i.e. $\Gamma = \{0, 0.5, 1, 1.5, 2, 2.5, 3\}$ dB. In a mini-batch, each channel SNR occupies the same proportion of the training data. In practice, the operational range of SNR may not be known a priori, and hence training SNR and operation SNR may have a mismatch. Thus, in this section, we focus on training the network with data generated under a *single* SNR and investigate how robust the BP-CNN decoder is under a range of SNRs.

It is worth noting that in the previous results, we have already reported some results with mismatched SNRs. For example, when $\eta = 0.8$, the channel SNR range for evaluating the performance, i.e. $-3 \sim 1$ dB, is already different from the SNR range for training data. This is because the strong correlation exists in the noise and CNN can achieve significant performance gains, and we have tested the BER performance under channel SNRs that are lower than the training SNRs. In this section, we report a more complete set of results.

In order to analyze how the data generation influences the system performance, we change the channel SNR range in the following experiments. Specifically, when $\eta = 0.8$ we conduct the experiments under three different settings: $\Gamma = \{-3, -2.5, -2, -1.5, -1, -0.5, 0\}$ dB, $\Gamma = \{-3\}$ dB and $\Gamma = \{0\}$ dB. When $\eta = 0.5$, we consider $\Gamma = \{0, 0.5, 1, 1.5, 2, 2.5, 3\}$ dB, $\Gamma = \{0\}$ dB and $\Gamma = \{3\}$ dB. The decoder architecture is BP(5)-CNN-BP(5) and the CNN structure is $\{4; 9, 3, 3, 15; 64, 32, 16, 1\}$. These settings are summarized in Table III.

The results are reported in Fig. 12. Generally speaking, generating training data under a wide range of channel

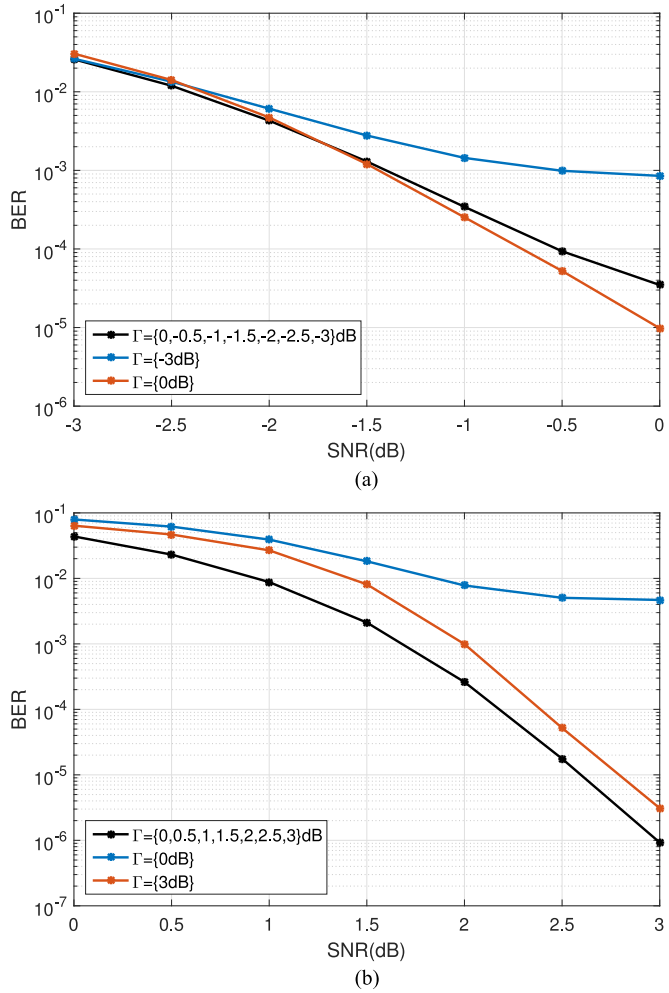


Fig. 12. Analyze the system robustness to the training data generated under different channel conditions. (a) $\eta = 0.8$, strong correlation. $\lambda = 0.1$. (b) $\eta = 0.5$, moderate correlation. $\lambda = 10$.

SNRs is a better choice. As shown in Fig. 12(b), training the network for 0 dB or 3 dB yields a worse performance than $\{0, 0.5, 1, 1.5, 2, 2.5, 3\}$ dB. This is mainly ascribed to deep learning technologies which depend on training data to select network parameters. When $\Gamma = \{3\}$ dB, the SNR is good and there are few errors in the input to CNN, i.e. $\hat{\mathbf{n}}$. With these "skewed" data, it is difficult for the network to learn how to remove errors. On the other hand, it is also not a good choice to generate training data under a bad channel condition, such as $\Gamma = \{0\}$ dB in Fig. 12(b). In this case, the training data contains too many errors, which is also not beneficial for the network to learn robust features of the channel noise.

Similar results are also obtained for $\eta = 0.8$ as shown in Fig. 12(a). It is worth noting that generating data under 0 dB provides a similar performance as generating data under -3 to 0 dB. It indicates that generating data under 0 dB is also a good choice when $\eta = 0.8$. Ultimately, how to select the *optimal* channel conditions for data generation is a difficult task. Therefore, we suggest that the training data be generated under multiple channel conditions to enhance the data diversity.

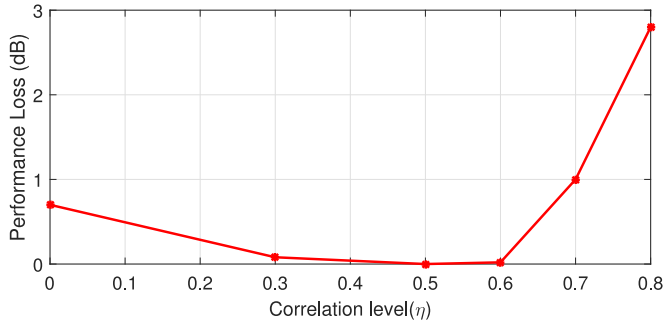


Fig. 13. The performance loss of the model trained for $\eta = 0.5$ under different η values.

E. Robustness to Mismatched Channel Correlation Models

The actual channel encountered in the "online" phase may be different from the model for which the network is trained during the "offline" phase. It is thus of great interest to study the impact of model mismatches to the proposed BP-CNN architecture. We have carried out some experiments to study how the proposed architecture performs when different mismatch happens. Specifically, we test the performance of the model trained for $\eta = 0.5$ under other η values, and compare its performance to the model specifically trained for each η (i.e., no mismatch). Fig. 13 reports the performance loss caused by the correlation mismatch. The tested system structure is BP(25)-CNN-BP(25) and the CNN is trained with the enhanced BP-CNN strategy. For $\eta = 0, 0.3, 0.5, 0.6, 0.7, 0.8$, the λ values used for obtaining the BP-CNN architecture under the matching channel model are 10, 10, 10, 0.5, 0.1, and 0.1 respectively. We can see that a small to moderate model mismatch does not cause a significant performance degradation. With a large mismatch, the performance loss also increases, particularly when η is large. However, we note that such performance loss is with respect to the BP-CNN architecture trained under the matching model. The mismatched BP-CNN architecture still outperforms the standard BP when η is large.[2]

F. Performance Comparison of CNN and BCJR Under Noise Correlation

The problem of channel decoding under colored noise can also be addressed by an intersymbol interference (ISI) channel approximation and applying the BCJR algorithm [42]. This approach first estimates the noise correlation, and then designs a filter to pre-process the received signal $\mathbf{y}$ before invoking the BP decoder, which is described as follows. We consider a filter $\mathbf{h}$ of the general form: $\mathbf{h} = [h_{-T}, h_{-T+1}, \ldots, h_0, \ldots, h_{T-1}, h_T]^T$ of length $2T + 1$. Then we have

$$y_i^{(c)} = \sum_{k=-T}^{T} \left(h_k s_{i+k} + h_k n_{i+k} \right), \tag{26}$$

[2]At $\eta = 0.8$, a performance gain of 0.7 dB compared to the standard BP is observed for the mismatched BP-CNN decoder.

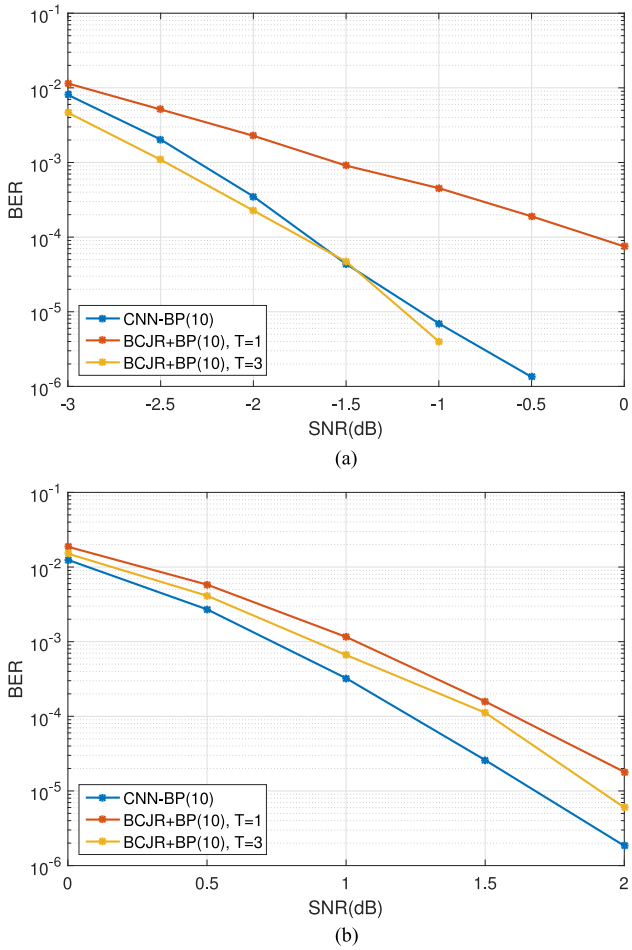


Fig. 14. Compare the performance of BCJR and CNN to handle noise correlation. (a) $\eta = 0.8$ (b) $\eta = 0.5$.

where $y_i^{(c)}$ denotes symbol i after convolution. Model (26) can be viewed as an ISI channel, and thus the LLR value of s_i can be estimated via applying the celebrated BCJR algorithm [43], by viewing the filtered noise as AWGN. The estimated LLR values are then fed to a standard BP decoder to obtain the source bits. This method is denoted as *BCJR-BP*.

The performance of BCJR-BP is largely influenced by the selection of the filter $\mathbf{h}$. We use the least-square criterion to design the filter, i.e., to minimize $\mathbb{E}(y_i^{(c)} - s_i)^2$. This problem can be easily solved and the optimal filter is given by

$$\mathbf{h} = (\mathbf{R} + P\mathbf{I})^{-1} \mathbf{1}_{T+1}, \tag{27}$$

where $\mathbf{R}$ is the estimated $(2T+1) \times (2T+1)$ covariance matrix of the channel noise, $\mathbf{I}$ is a $(2T+1) \times (2T+1)$ identity matrix, and $\mathbf{1}_{T+1}$ is a $(2T+1) \times 1$ vector with the $(T+1)$th element being 1 and all other elements being 0.

In order to fairly compare the performance of BCJR and CNN in terms of how they handle noise correlation, we have removed the first BP decoding process in our proposed method and implemented a revised structure CNN-BP for the numerical comparison.[3] As mentioned in Section III-F, this is a special form of the proposed architecture. The baseline strategy is used for CNN training. Ten BP iterations are set for both CNN-BP and BCJR-BP. The simulation results are shown in Fig. 14 for both $\eta = 0.8$ and $\eta = 0.5$. We see that CNN-BP can achieve about 0.25 dB performance gain over BCJR-BP ($T = 3$) at BER $= 10^{-4}$ for $\eta = 0.5$. When $\eta = 0.8$, CNN and BCJR have similar performance.

We further comment that the CNN-based approach has some practical advantages over BCJR-BP. First, BCJR consists of both forward and backward recursions, making it difficult for parallel implementation which is an important advantage of CNN. Second, noting that the complexity of the BCJR algorithm grows exponentially with the memory length, i.e., $2T+1$, the value of T cannot be very large in practice. This will increase the model mismatch when the channel memory is long. On the other hand, the complexity of CNN is linear in the filter length, and it is more suitable for channels with long memory.

[3]BCJR-BP does not have a first decoding process before applying BCJR.

V. Conclusions and Future Directions

In this paper, we have designed a novel iterative BP-CNN decoding architecture to handle correlated channel noise. The proposed framework serially concatenates a CNN with a BP decoder, and iterates between them. The BP decoder is to estimate the coded bits and indirectly estimate the channel noise. The CNN is to remove the channel noise estimation errors of the BP decoder by learning the noise correlation. To implement the framework, we have proposed to adopt a fully convolutional network structure and provided two strategies to train the network. We have carried out extensive simulations to show the effectiveness of the proposed iterative BP-CNN decoder.

There are some important research directions for the iterative BP-CNN framework, which is worth investigating in the future. First, the loss function is not completely equivalent to the system performance measurement, and finding a better loss function is one of the important future directions. Furthermore, as mentioned in Section III-F, the iterative system proposed in this paper can be unfolded into an open-loop system, and for maximum flexibility, we can design different number of BP iterations in each BP decoding process as well as different CNN structures. This system can be denoted as BP(n_1)-CNN1-BP(n_2)-CNN2-...-BP(n_x)-CNNx-BP(n_{x+1}), and provides the maximum degrees of freedom as we can allocate the complexity in all components to maximize the performance. We plan to study this general architecture in a future work.

Furthermore, we note that the CNN adopted in our architecture is a pure *feed-forward* network without memory or state information. In the deep learning literature, there is another celebrated network structure, called *recurrent neural networks* (RNN), which is mainly used in language processing [44]. RNN is effective in taking advantage of the sequential information and it has potential to exploit the correlation among the noise sequence by adding a unit to record the network hidden state. The exploitation of the RNN structure with memory in conjunction with BP decoding is an important and interesting topic that we plan to investigate in the future.

REFERENCES

[1] R. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[2] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[3] D.-S. Shiu, G. J. Foschini, M. J. Gans, and J. M. Kahn, "Fading correlation and its effect on the capacity of multielement antenna systems," *IEEE Trans. Commun.*, vol. 48, no. 3, pp. 502–513, Mar. 2000.

[4] H. T. Hui, "Influence of antenna characteristics on MIMO systems with compact monopole arrays," *IEEE Antennas Wireless Propag. Lett.*, vol. 8, pp. 133–136, 2009.

[5] S. K. Sharma, S. Chatzinotas, and B. Ottersten, "SNR estimation for multi-dimensional cognitive receiver under correlated channel/noise," *IEEE Trans. Wireless Commun.*, vol. 12, no. 12, pp. 6392–6405, Dec. 2013.

[6] A. Hajimiri and T. H. Lee, "A general theory of phase noise in electrical oscillators," *IEEE J. Solid-State Circuits*, vol. 33, no. 2, pp. 179–194, Feb. 1998.

[7] L. Di Bert , P. Caldera, D. Schwingshackl, and A. M. Tonello, "On noise modeling for power line communications," in *Proc. IEEE Int. Symp. Power Line Commun. Appl.*, 2011, pp. 283–288.

[8] S. S. Tehrani, B. F. Cockburn, and S. Bates, "On the effects of colored noise on the performance of LDPC codes," in *Proc. IEEE Workshop Signal Process. Syst. Des. Implementation*, 2006, pp. 226–231.

[9] T. Souvignier, J. K. Wolf, and A. Dati, "Turbo decoding for partial response channels with colored noise," *IEEE Trans. Magn.*, vol. 35, no. 5, pp. 2322–2324, Sep. 1999.

[10] J. D. Wang and H. Y. Chung, "Trellis coded communication systems-colored noise and the swapping technique," *IEEE Trans. Commun.*, vol. 38, no. 9, pp. 1549–1556, Sep. 1990.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[13] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning affordance for direct perception in autonomous driving," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 2722–2730.

[14] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *Proc. 54th Annu. Allerton Conf. Commun., Control, Comput.*, Sep. 2016, pp. 341–346.

[15] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention," *IEEE Int. Symp. Signal Proc. Inf. Technol.*, pp. 223–228, 2016.

[16] T. J. O'Shea, S. Hitefield, and J. Corgan, "End-to-end radio traffic sequence recognition with recurrent neural networks," in *Proc. IEEE Global Conf. Signal Inf. Process.*, 2016, pp. 277–281.

[17] E. Nachmani, E. Marciano, D. Burshtein, and Y. Be'ery, "RNN decoding of linear block codes," arXiv:1702.07560. [Online]. Available: http://arxiv.org/abs/1702.07560

[18] L. Lugosch and W. J. Gross, "Neural offset min-sum decoding," *IEEE Int. Symp. Inf. Theory*, 2017.

[19] T. Gruber, S. Cammerer, J. Hoydis, and S. tenBrink, "On deep learning-based channel decoding," in *Proc. IEEE 51st Annu. Conf. Inf. Sci. Syst.*, 2017, pp. 1–6.

[20] S. Cammerer, T. Gruber, J. Hoydis, and S. ten Brink, "Scaling deep learning-based decoding of polar codes via partitioning," arXiv:1702.06901. [Online]. Available: http://arxiv.org/abs/1702.06901

[21] T. J. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Trans. Cogn. Commun. Netw.*, vol. 3, no. 4, pp. 563–575, 2017.

[22] N. Farsad and A. Goldsmith, "Detection algorithms for communication systems using deep learning," arXiv:1705.08044. [Online]. Available: http://arxiv.org/abs/1705.08044

[23] N. Samuel, T. Diskin, and A. Wiesel, "Deep MIMO detection," in *Proc. IEEE Int. Workshop Signal Process. Adv. Wireless Commun.*, 2017.

[24] S. Dörner, S. Cammerer, J. Hoydis, and S. ten Brink, "Deep learning-based communication over the air," *IEEE J. Select. Topics Signal Process.*, 2017.

[25] V. Jain and S. Seung, "Natural image denoising with convolutional networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 769–776.

[26] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 2, pp. 295–307, Feb. 2016.

[27] T. Thadewald and H. Büning, "Jarque–Bera test and its competitors for testing normality: A power comparison," *J. Appl. Statist.*, vol. 34, no. 1, pp. 87–105, 2007.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[29] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[30] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[31] I. Dimnik and Y. Be'ery, "Improved random redundant iterative HDPC decoding," *IEEE Trans. Commun.*, vol. 57, no. 7, pp. 1982–1985, Jul. 2009.

[32] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising," *IEEE Trans. Image Process.*, vol. 26, no. 7, pp. 3142–3155, Jul. 2017.

[33] S. Lin and D. Costello, *Error Control Coding*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2004.

[34] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010, pp. 807–814.

[35] X.-J. Mao, C. Shen, and Y.-B. Yang, "Image restoration using very deep convolutional encoderdecoder networks with symmetric skip connections," *Advances Neural Inf. Process. Syst.*, pp. 2802–2810, 2016.

[36] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[37] M. Helmling and S. Scholl, "Database of channel codes and ML simulation results," Wimax 3/4A, University of Kaiserslautern, Kaiserslautern, Germany, 2016. [Online]. Available: www.uni-kl.de/channel-codes

[38] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Des. Implementation.* Savannah, GA, USA, 2016.

[39] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. Int. Conf. Artif. Intell. Statist.*, vol. 9, 2010, pp. 249–256.

[40] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represen.*, 2015.

[41] Q. You, J. Luo, H. Jin, and J. Yang, "Robust image sentiment analysis using progressively trained and domain transferred deep networks," in *Proc. AAAI Conf. Artif. Intell.*, 2015, pp. 381–388.

[42] D. J. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2003.

[43] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Trans. Inf. Theory*, vol. IT-20, no. 2, pp. 284–287, Mar. 1974.

[44] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. INTERSPEECH*, vol. 2, 2010, pp. 1045–1048.

Fei Liang received the B.Sc. degree in electrical engineering and information science in 2013 from the University of Science and Technology of China, Hefei, China, where he is currently working toward the Ph.D. degree in signal and information processing. His research interests include multimedia communication, digital communication and intelligent communication.

Cong Shen (S'01–M'09–SM'15) received the B.S. and M.S. degrees from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2002 and 2004, respectively, and the Ph.D. degree from the Department of Electrical Engineering, University of California, Los Angeles, Los Angeles, CA, USA, in 2009. From 2009 to 2014, he was with Qualcomm Research, San Diego, CA, USA, where he focused on various cutting-edge research topics including cognitive radio, TV white space, and heterogeneous and ultradense networks. In 2015, he returned to academia and joined the University of Science and Technology of China (USTC) as the 100 Talents Program Professor in the School of Information Science and Technology. His general research interests include communication theory, wireless networks, and machine learning. He is currently an Editor for the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS.

Feng Wu (M'99–SM'06–F'13) received the B.S. degree in electrical engineering from Xidian University, Xi'an, China, in 1992, and the M.S. and Ph.D. degrees in computer science from the Harbin Institute of Technology, Harbin, China, in 1996 and 1999, respectively.

He was a Principle Researcher and Research Manager with Microsoft Research Asia, Beijing, China. He is currently a Professor and the Dean of the School of Information Science and Technology, University of Science and Technology of China, Hefei, China. He has authored or coauthored more than 300 high-quality papers, including several dozens of the IEEE Transactions papers and top conference papers on MOBICOM, SIGIR, CVPR, and ACM MM. He has 77 granted U.S. patents. His 15 techniques have been adopted into international video coding standards. His research interests include image and video compression, media communication, and media analysis and synthesis.

Dr. Wu serves or had served as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEM FOR VIDEO TECHNOLOGY, the IEEE TRANSACTIONS ON IMAGE PROCESSING, the IEEE TRANSACTIONS ON MULTIMEDIA, and several other international journals. He was the TPC Chair of MMSP 2011, VCIP 2010, and PCM 2009, and the Special Sessions Chair of ICME 2010 and ISCAS 2013. He was the recipient of the Best Paper Awards of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY 2009, PCM 2008, and VCIP 2007, as well as the IEEE Circuits and Systems Society 2012 Best Associate Editor Award.