Received January 7, 2022, accepted January 20, 2022, date of publication January 26, 2022, date of current version February 3, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3146336

Two Novel Semi-/Auto-Adaptive SNR Algorithms to Efficiently Train Deep Neural SPA Decoders

CHUN-MING HUANG
Department of Electronic Engineering, National Formosa University, Huwei, Yunlin 632301, Taiwan
e-mail: huangcm@nfu.edu.tw

This work was supported by the Ministry of Science and Technology, Taiwan, under Grant MOST 107-2218-E-150-008-MY2.

ABSTRACT In the past few years, deep learning has been widely used in various fields due to its outstanding progress. One of the latest applications of deep learning is to use a neural network (NN) with trainable multiplicative weights to design decoders for error-correcting codes. High quality data are essential for deep learning to train robust NN models. In this study, two novel semi-/auto-adaptive SNR algorithms are proposed to efficiently train the neural decoders based on the Sum-Product Algorithm (SPA). For illustration, several neural SPA decoders for the Bose-Chaudhuri-Hocquenghem (BCH) code and low-density parity-check (LDPC) code have been constructed as examples. Simulation results show that, compared with the original neural decoders, the performance of these neural decoders trained by the proposed algorithms can be improved in the range of 0.2 to 0.6 dB. Moreover, the training time required for these decoders to achieve convergence can be reduced by up to 28.8% for the BCH code, and up to 35.6% for the LDPC code, without increasing decoding complexity.

INDEX TERMS Neural network (NN), sum-product algorithm (SPA), Bose-Chaudhuri-Hocquenghem (BCH) code, low-density parity-check (LDPC), semi-/auto-adaptive SNR algorithm.

I. INTRODUCTION

Owing to the presence of noise interference in digital message transmission, the received message may not be exactly the same as what was sent. To achieve reliable communication, it is necessary to detect and correct the errors by using error correcting codes (ECCs). In the past few decades, several near-capacity ECCs have been proposed for error-free data transmission over noisy channels, such as the turbo codes [1], low-density parity-check (LDPC) codes [2], and polar codes [3]. Shao *et al.* provided an overview and comparison of the turbo code, LDPC code, and polar code in [4]. Detailed descriptions of the capabilities of these three codes are given for meeting different requirements associated with the enhanced Mobile Broad Band (eMBB), Ultra-Reliable Low Latency Communication (URLLC), and massive Machine Type Communication (mMTC) applications of 5G, as well as the application specific integrated circuit (ASIC) implementation of the decoders. Among these codes, LDPC codes have attracted widespread attention through the excellent performance of the iterative belief propagation (BP) decoding algorithm [2]. Because of its widespread popularity, adaptability, and parallelism in cost-effective hardware implementations, LDPC codes have been used to improve data reliability in various communication applications [5].

The BP decoding algorithms are widely preferred in many wireless communication standards because of their excellent error rate performance. Because the BP decoding algorithms involve great number of logarithmic and multiplicative operations when updating the check node, the high computational complexity burdens the hardware. In an effort to reduce the computational complexity, Fossorier *et al.* introduced min-sum algorithm (MSA) for fast iterative decoding of LDPC codes [6]. However, MSA has significant performance degradation, because the computation of the check-to-variable message is simplified to a minimum operation instead of a hyperbolic tan calculation. Chen and Fossorier [7] introduced the Normalized Min-Sum (NMS) and Offset Min-Sum (OMS) algorithms to modify the MS algorithm to achieve better performance. Numerical results showed that with one properly chosen parameter for each of these two algorithms, performances can be close to that of the BP algorithm. In [8], one simple method with less complex arithmetic operations to find the offset correction factor had been demonstrated. Chang *et al.* [9] proposed a conditional

The associate editor coordinating the review of this manuscript and approving it for publication was Massimo Cafaro.

variable node (VN) selecting metric to realize informed dynamic scheduling (IDS) LDPC decoding schedules. The goal is to find VNs with probable incorrect decisions and correct the errors by updating them. To reduce decoding latency, multi-edge updating versions of algorithms are realized by increasing the degrees of parallelism of the update. In [10], Roberts *et al.* conducted a comprehensive review of various LDPC decoding algorithms based on their working principles, error correcting capabilities, and computational complexity. However, the BP decoding algorithm may be not suitable for high density parity check (HDPC) codes, such as Reed Solomon (RS) codes and Bose-Chaudhuri-Hocquenghem (BCH) codes. Because these HDPC codes have a large number of short cycles in the graph, there will be correlation between the messages, which results in error propagation.

In recent years, deep learning has demonstrated tremendous progress in various tasks, such as machine translation, autonomous vehicles, and object recognition. Serval researchers have presented methods of applying deep learning to wireless communication systems. Almohamad *et al.* proposed to sample the received signals and generate the corresponding asynchronous amplitude histograms (AAHs) [11] and two-dimensional asynchronously sampled in-phase-quadrature amplitudes' histograms (2D-ASIQHs)-based images [12]. Due to the unique statistical properties of the AAHs, they can be used to obtain the information about different signal parameters (i.e., modulation type and SNR). Next, the significant features of these AAHs (or 2D-ASIQHs images) are extracted through Principal components analysis (PCA) and used to train the Support Vector Machine (SVM) based classifier and regressor. Simulation results show that this SVM-based system has a good accuracy rate in identifying and predicting the modulation type and SNR value of the received signal. Obtaining the modulation type and SNR information in advance will enable the receiver to make corresponding adjustments to improve system performance, which is crucial for the receiver design in future wireless communication systems. Nachmani *et al.* [13], [14] proposed one deep neural network (DNN) architecture for decoding HDPC codes. Because the edges of the Tanner graph are assigned with trainable weights and biases, this DNN decoder can be regarded as a weighted-BP (WBP) decoder. From the simulation, it can be seen that the performance can be improved by optimally training the neural network parameters. Gruber *et al.* [15] proposed to use a fully-connected (FC) neural network (NN) to decode the random codes and structure codes. When the code length was very short, the simulation results showed that this NN decoder performed as well as the maximum a posterior (MAP) decoder. Kim *et al.* [16] demonstrated that trained recurrent neural network (RNN) architectures can decode convolutional and turbo codes with close to optimal performance under the additive white Gaussian noise (AWGN) channel. Lugosch and Gross [17] proposed to apply deep learning technology to the offset min-sum algorithm (OMS) to achieve similar decoding performance as that in [13], while requiring fewer trainable parameters and lower hardware complexity. Wang *et al.* [18] adopted model-driven deep learning and proposed shared neural normalized min-sum (SNNMS) decoding to reduce the number of correction factors and lower the complexity. Vasić *et al.* [19] utilized DNNs to MSA for decoding LDPC codes. The simulation results showed that the neural decoder can perform no worse than conventional MSA. Chu *et al.* [20] proposed a neural-network optimized low-resolution decoding (NOLD) algorithm to improve the performance degradation of the MSA incurred by low-resolution quantization. In [21], we applied the DNN architecture to the Sum-Product Algorithm (SPA) decoder to improve the performance of the optical code-division multiple-access (OCDMA) system with LDPC codes.

Generating a suitable training dataset under various signal-to-noise (SNR) ranges to train a robust neural network model is an interesting and important issue. In [14], Nachmani *et al.* demonstrated that different decoding performances on the same validation set were obtained while training WBP models with varying SNR ranges. Gruber *et al.* [15] introduced a metric to compare the training performance of an NN decoder having a particular SNR with that of an MAP decoder over a range of SNR values. If the training set includes the entire codebook, for both structure and random codes, training the NN decoders with a certain SNR value can generalize input values with arbitrary SNR. Otherwise, only the NN decoder used for structure codes would be capable of generalizing unknown codewords. In [22], Lian *et al.* utilized parameter adapter networks (PANs) to establish the relationship between the SNR and WBP parameters. Several shallow NNs were used to estimate the SNR-dependency for each parameter in the decoding algorithm. In [23], Be'Ery *et al.* proposed two supervised learning methods, which actively sampled the training data through hamming distance and reliability parameters, respectively. It was shown that this active sampling method provides an efficient way to separate useful training data.

As the method of finding useful data is a key factor for training the NN, we propose two novel semi-/auto-adaptive SNR algorithms without increasing the complexity of the decoder. These algorithms can be applied to the aforementioned DNN decoders, such as the neural SPA (NSPA), neural OMS (NOMS), and SNNMS. For clarity of exposition, we take the NSPA used in [13], [14], [21] as an example in this work. Intuitively, the BCH codes in [13], [14] and LDPC codes in [21] are also used to verify the effectiveness of the proposed algorithms. In addition, inspired by [13], [14], two modified NSPA (MNSPA) decoders are investigated and demonstrated. From the simulation results, we can observe that the performance of these neural decoders can be further improved using the proposed adaptive SNR algorithms.

The remainder of this paper is organized as follows. Section II introduces the necessary background for the neural decoder. In Section III, two novel semi-/auto-adaptive SNR algorithms are described in detail. Section IV presents the simulation results, and Section V concludes the paper.

II. PRELIMINARIES

Consider one (N, K) linear block code with a parity-check matrix $\boldsymbol{H}$, where N is the code length and K is the message length. Let $\mathrm{x} = (x_1, x_2, \ldots, x_N)$ be the codeword transmitted over the AWGN channel with common binary phase-shift keying (BPSK) mapping to $\{\pm 1\}$ and $\mathrm{y} = (y_1, y_2, \ldots, y_N)$ be the corresponding received output vector, where $y_v = (-1)^{x_v} + n_v$ for $1 \le v \le N$, and n_v is Gaussian noise with standard deviation σ. The signal-to-noise ratio per bit used in this work can be represented as $\mathrm{SNR} = E_s/(RN_0) = 1/(2\sigma^2)$, where E_s denotes the signal energy, R is the code rate of the linear block code and N_0 is the power spectral density of noise.

Thus, the initial log-likelihood ratio (LLR) of the vth received value can be calculated as

$$L_v = \log \frac{\Pr(x_v = 0|y_v)}{\Pr(x_v = 1|y_v)} = \frac{2y_v}{\sigma^2} \tag{1}$$

A. SUM-PRODUCT ALGORITHM (SPA) DECODER

First, we review the sum-production algorithm, which is an iterative computing algorithm for decoding the LDPC code. For each iteration, the posterior LLRs of the received vector are computed by exchanging the messages between processing nodes. The symbols and notations used in SPA are listed as follows:

- E: the number of edges in the Tanner graph.
- T: the maximum iteration number in the SPA.
- $M(c)$: The set of variable nodes connected to check node c.
- $M(c)\backslash v$: The set of variable nodes connected to check node c, *not including* variable node v.
- $N(v)$: The set of check nodes connected to variable node v.
- $N(v)\backslash c$: The set of check nodes connected to variable node v, *not including* check node c.
- $L^k_{(a,b)}$: The LLR message from node a to node b in hidden layer k.

At the i-th iteration, the messages from variable nodes to check nodes are computed by:

$$L^i_{(v,c)} = L_v + \sum_{c' \in N(v)\backslash c} L^{i-1}_{(c',v)} \tag{2}$$

Notice that $L^0_{(c',v)} = 0$ due to no information being present at the check nodes.

The messages from check nodes to variable nodes are updated by:

$$L^i_{(c,v)} = 2\tanh^{-1}\left(\prod_{v' \in M(c)\backslash v} \tanh\left(\frac{L^{i-1}_{(v',c)}}{2}\right)\right) \tag{3}$$

The final output variable node value is calculated by:

$$\hat{x}_v = L_v + \sum_{c' \in N(v)} L^{2T}_{(c',v)} \tag{4}$$

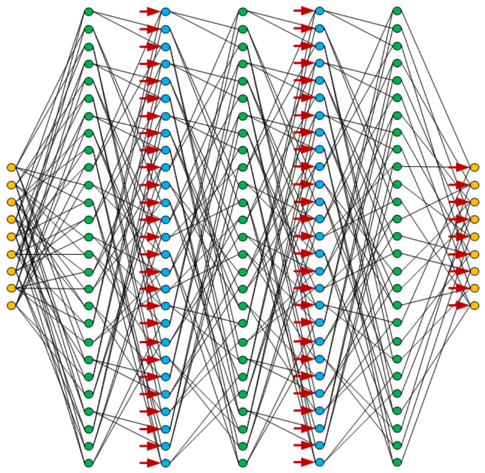

FIGURE 1. The NN architecture for the (9, 2) linear block code.

At the end of each iteration, the decoded codeword can be obtained by:

$$\hat{y}_v = \begin{cases} 1, & \mathit{if}\ \hat{x}_v < 0 \\ 0, & \mathit{else}, \end{cases} \quad \text{for } 1 \le v \le N \tag{5}$$

Procedure stops if $\hat{y}\,\boldsymbol{H}^T = 0$ or a maximum number of iteration has been reached.

B. DEEP NEURAL NETWORK

Next, we introduce the structure of these DNN decoders, whose network architectures are non-fully connected neural networks. Essentially, this can be viewed as a Trellis representation of the Tanner graph, which extends two hidden layers from one iteration of the SPA.

The decoder is composed of one input layer of size N, $2T$ hidden layers of size E, and one output layer of size N. For each hidden layers, each neuron in that layer indicates the message transmitted over an edge in the Tanner graph. For illustration, one 9×9 parity-check matrix $\boldsymbol{H}$ is constructed as follows:

$$H = \begin{bmatrix} 1&0&0&1&0&0&1&0&0 \\ 0&1&0&0&1&0&0&1&0 \\ 0&0&1&0&0&1&0&0&1 \\ 1&0&0&0&1&0&0&0&1 \\ 0&1&0&0&0&1&1&0&0 \\ 0&0&1&1&0&0&0&1&0 \\ 1&0&0&0&0&1&0&1&0 \\ 0&1&0&1&0&0&0&0&1 \\ 0&0&1&0&1&0&1&0&0 \end{bmatrix} \tag{6}$$

By using Gaussian elimination, we can know that $\boldsymbol{H}$ has a rank of 7, which means that it can be considered as a parity-check matrix of (9, 2) linear block code. The code length N is 9, and the column and row weights are both 3. Assume that the maximum iteration number T is 3, the corresponding decoder has one input layer, six hidden layers, and one output layer. The first (last) layers have 9 input (output) nodes, and all six hidden layers have 27 neurons corresponding to the 27 edges over its Tanner graph, as shown in Fig. 1.

The connections among these layers are determined as follow:

1. For the first (last) layer, neuron t is connected to a single input node x_v in the input (output) layer if x_v is incident to edge t, where $1 \leq v \leq 9$ and $1 \leq t \leq 27$.
2. For the remaining odd (or even) hidden layers, neuron t in this layer is connected to the neurons in the previous layer, whose corresponding edges in the Tanner graph are incident to $N(v)\backslash c$ (or $M(c)\backslash v$).

Note that since there is no information on the check nodes after the first initialization, the first and second hidden layers can be merged together. The self LLR messages L_v are plotted as small arrows in Fig. 1.

C. NSPA DECODER

For the NSPA decoder [13], only the variable-to-check messages (i.e., odd hidden layers) and the output node are assigned with trainable weights. For odd hidden layer i, the messages are calculated as:

$$L^i_{(v,c)} = \tanh\left(\frac{1}{2}\left(w^i_v L_v + \sum_{c' \in N(v)\backslash c} w^i_{c',v,c} L^{i-1}_{(c',v)}\right)\right) \qquad (7)$$

where w^i_v and $w^i_{c',v,c}$ are trainable weights assigned to LLR messages L_v and $L^{i-1}_{(c',v)}$, respectively. As stated previously, $L^0_{(c',v)} = 0$ due to no information being present at the check nodes.

For even hidden layer i, the check-to-variable messages are updated as:

$$L^i_{(c,v)} = 2\tanh^{-1}\left(\prod_{v' \in M(c)\backslash v} L^{i-1}_{(v',c)}\right) \qquad (8)$$

The output marginalization is computed as

$$\hat{x}_v = \sigma\left(w^{2T+1}_{v,out} L_v + \sum_{c' \in N(v)} w^{2T+1}_{c',v,c,out} L^{2T}_{(c',v)}\right) \qquad (9)$$

where $\sigma(x) = (1 + e^{-x})^{-1}$ is a sigmoid function to convert the value to the range [0, 1]. If the output value is in the range [0, 0.5], it is determined to be bit 1, otherwise, bit 0.

D. MNSPA DECODER

To reduce the number of trainable weights, two MNSPA decoder are investigated as follows.

1) TYPE I

Inspired by [14], one simple method to reduce the trainable weights of the NSPA can be easily implemented as follows. For odd hidden layer i, the variable-to-check messages are updated as:

$$L^i_{(v,c)} = \tanh\left(\frac{1}{2}\left(L_v + \sum_{c' \in N(v)\backslash c} L^{i-1}_{(c',v)}\right)\right) \qquad (10)$$

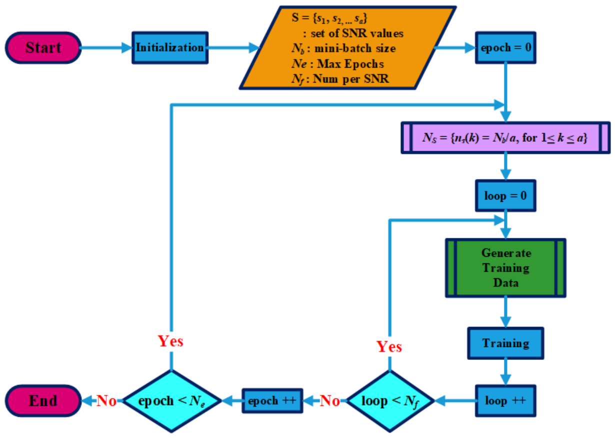

FIGURE 2. The training flowchart of the DNN decoders.

Notice that the main difference between (7) and (10) is that the trainable weights are removed in (10). Similarly, $L^0_{(c',v)} = 0$ as no information is present at the check nodes in the beginning.

For even hidden layer i, the check-to-variable messages are calculated as:

$$L^i_{(c,v)} = w^i_{c,v} \cdot \left(2\tanh^{-1}\left(\prod_{v' \in M(c)\backslash v} L^{i-1}_{(v',c)}\right)\right) \qquad (11)$$

where $w^i_{c,v}$ is the learnable weight parameter for the edge connecting check node c to variable node v. Note that the input for the $\tanh^{-1}$ function is clipped in the range of −0.999 to 0.999 to stabilize the operation of the decoder.

Finally, the output marginalization is calculated as

$$\hat{x}_v = \sigma\left(\mathbf{W}_{v,out}(v) \cdot L_v + \mathbf{W}_{cv,out}(v) \cdot \sum_{c' \in N(v)} L^{2T}_{(c',v)}\right) \qquad (12)$$

where $\mathbf{W}_{v,out}(v)$ and $\mathbf{W}_{cv,out}(v)$ are the learnable weights for the self LLR message L_v and the message from the check node s to the variable node x_v, respectively. The output bit is determined to be 0 if the value is in the range (0.5, 1], otherwise 1.

2) TYPE II

Similar to [22], the MNSPA Type II decoder can be viewed as the NSPA decoder with simple scaling trainable weights. The equations used to calculate the variable-to-check messages for odd hidden layer i and output marginalization are the same as (10) and (12), respectively. For even hidden layer i, the equation for updating the check-to-variable messages is modified as:

$$L^i_{(c,v)} = w^i \cdot \left(2\tanh^{-1}\left(\prod_{v' \in M(c)\backslash v} L^{i-1}_{(v',c)}\right)\right) \qquad (13)$$

where w^i is the learnable weight parameter for the edges connecting to hidden layer i.

For brevity, the two types of MNPSA decoders are named MNPSA-I and MNPSA-II decoders for the remainder of this paper. The main differences between the MNSPA decoders and the NSPA decoder are summarized as follows:

1. **NSPA decoder:** Four types of weight matrices $\{[w_v^i]$, $[w_{c',v,c}^i]$, $[w_{v,out}^{2T+1}]$, $[w_{c',v,c,out}^{2T+1}]\}$ are used for training. The first two matrices are assigned to the edges of odd hidden layer i (i.e., the variable-to-check messages), and the remaining two matrices are assigned to the final output marginalization. The sizes of these matrices are $\{N \times E, E \times E, 1 \times N, E \times N\}$, in sequence. Assume that the row weight of $\boldsymbol{H}$ is w_r; the NSPA decoder must train $[N+E(w_r - 1)]T + N + NE$ weights.
2. **MNSPA-I decoder:** As discussed above, three types of weight matrices $\{[w_{c,v}^i]$, $\mathbf{W}_{v,out}$, $\mathbf{W}_{cv,out}\}$ are used for training, but the sizes of these matrices are reduced to $\{1 \times E,\ 1 \times N,\ 1 \times N\}$, respectively. The MNSPA-I decoder must train $ET + 2N$ weights, and the training process and computation complexity can be greatly reduced.
3. **MNSPA-II decoder:** According to (12) and (13), two types of weight matrices $\{\mathbf{W}_{v,out}, \mathbf{W}_{cv,out}\}$ of size $\{1 \times N,\ 1 \times N\}$ and one trainable weight w^i are assigned to the output layer and each even hidden layer i, respectively. Only $T + 2N$ weights need to be trained, thus, the training process and computation complexity of the MNSPA-II decoder is the lowest among the three decoders.

The performances of the decoders are independent of the transmitted codewords because the channel output is symmetric. Thus, the training database is constructed by using the zero codeword with noise. The goal is to train the decoder to recognize the output as close as possible to the zero codeword.

The neural network is trained by using an optimization process, which requires a loss function to calculate the model error. In this study, the binary cross-entropy (BCE) loss function is used for training as follows:

$$L_{BCE} = -\frac{1}{N}\sum_{v=1}^{N} x_v \log\left(\hat{x}_v\right) + (1 - x_v)\log\left(1 - \hat{x}_v\right) \quad (14)$$

where x_v and $\hat{x}_v$ are the vth element of the transmitted codeword and the neural decoder output, respectively.

III. SEMI-/AUTO-ADAPTIVE SNR ALGORITHMS

The NSPA, MNSPA-I, and MNSPA-II decoders are trained and optimized by minimizing the BCE loss function (14) using stochastic gradient descent (SGD) with mini-batch. Assume that the mini-batch size is N_b, the size of each training dataset can be represented as $N_b \times N$. Therefore, without loss of generality, we can derive the average mini-batch BCE (MBBCE) loss function as follow:

$$L_{MBBCE} = \frac{1}{N_b}\sum_{j=1}^{N_b} L_{BCE}\left(j\right) \quad (15)$$

where $L_{BCE}(j)$ represents the BCE value calculated by substituting row j of the mini-batch training dataset.

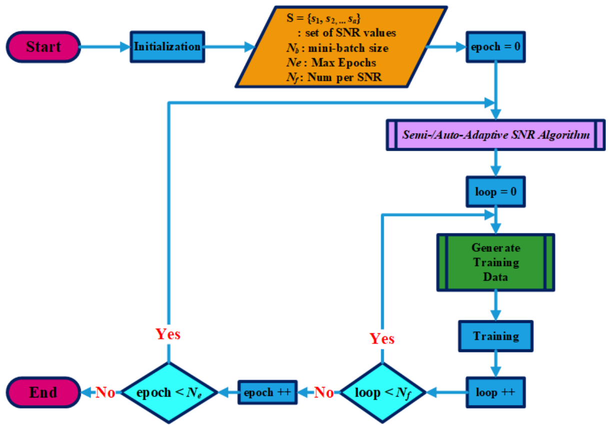

FIGURE 3. The training flowchart of the DNN decoders with semi-/auto-adaptive SNR algorithms.

The mini-batch training dataset consists of a range of different SNRs. It is clear that the variance of noise increases (or decreases) when the SNR decreases (or increases). If the SNR is close to zero, the noise is too large for the NN to learn the code structure. Similarly, if the SNR is close to infinity, it is also redundant for training the NN to handle the noise. Generating a suitable dataset under various SNR ranges is one key factor for training the NN. This inspired us to determine how to adjust the number of codewords with varying SNR in one epoch. First, the symbols and notations used in the algorithms are defined as follows:

- Let a be the number of SNR values used for generating the training dataset, and $S = \{s_k$, for $1 \le k \le a\}$ denotes the set of training SNR values.
- $N_s = \{n_s(k)$, for $1 \le k \le a\}$ is the set of the number of codeword of each SNR.
- $P = \{p_k = n_s(k)/N_b$, for $1 \le k \le a\}$ is the set of SNR ratios.
- N_e: the number of maximum epochs.
- N_f: the samples for each SNR.

Fig. 2 illustrates the flowchart of the DNN decoders. Generally, the mini-batch training dataset is constructed by generating codewords with uniform distribution SNRs (i.e., $N_s = \{n_s(k) = N_b/a$, for $1 \le k \le a\}$). For clarity, assume that the mini-batch size is 128, and the number of training SNRs is 8 (i.e., from 1 dB to 8dB). Then, the number of codewords for each SNR $n_s(k)$ is 16, and the p_k is 1/8 for $1 \le k \le 8$.

Because these decoders are optimized by minimizing the loss function, one intuitive idea to realize the SNR adaption is to modify the MBBCE equation as follows:

$$\begin{aligned} L_{MBBCE} &= \frac{1}{N_b}\sum_{k=1}^{a}\sum_{l=1}^{n_s(k)} L_{BCE}\left(l\right) \\ &= \sum_{k=1}^{a}\frac{n_s\left(k\right)}{N_b}\left(\frac{1}{n_s\left(k\right)}\sum_{l=1}^{n_s(k)} L_{BCE}\left(l\right)\right) \\ &= \sum_{k=1}^{a} p_k L_{SNR}\left(k\right) \end{aligned} \quad (16)$$

where $L_{SNR}(k)$ represents the MBBCE value when SNR is k. From (16), it can be observed that the MBBCE is modified to calculate the value of each SNR, rather than the summation of all SNRs in (15). Thus, we can utilize the set of SNR ratios P to adjust the number of codewords of different SNRs in one epoch.

The flowchart of the proposed algorithms is shown in Fig. 3. The processes for initialization and input are the same as the original flowchart in Fig. 2. The main difference is that the proposed algorithms will determine whether to adjust the set P in each epoch based on the optimization of MBBCE. Next, these two SNR algorithms are described in detail.

A. SEMI-ADAPTIVE SNR ALGORITHM

In the following, the Semi-Adaptive SNR algorithm is described in detail:

1) INITIALIZATION

- Set the values of a and N_b, then, the sets of S, N_s, and P can be determined.
- Set the value of $f_{optimal}$ and f_{att}, where the former denotes the desired optimal factor for the MBBCE value, and the latter represents the desired attenuation rate for SNR adaptation. Note that $0 < f_{optimal}, f_{att} \leq 1.0$.
- Let $L_{pre} = 0$, where L_{pre} denotes the MBBCE value for the previous epoch.
- Let $L_{cur} = 0$, where L_{cur} denotes the MBBCE value for the current epoch.
- Set $T_{att} = 0$, where T_{att} is the attenuation time of the SNR.

2) TRAINING

As shown in Figs. 2 and 3, the value of N_s used for generating the training data in one epoch is the same (i.e., if loop $< N_f$). The value of MBBCE is calculated by using (16) and then used in algorithm 1 to process the SNR adaptation. If the MBBCE value of the current epoch does not reach the desired optimal value (line 2), algorithm 1 will set the counter as a trigger condition to adaptively adjust the SNR ratios and adjust the number of codewords for each SNR (lines 5-7).

The detailed adaptive adjustment process is summarized in Algorithm 2. Because it is almost impossible to recover the signal if the noise is large, we propose to sequentially attenuate the codeword number of small SNRs. If the value of T_{att} is larger than zero, we multiply the SNR ratio p_i by $(1 - f_{att})$ to the power of (T_{att} - i), where $1 \leq i \leq a$ (lines 6-13). Next, the set of SNR ratios will be normalized and then used to adjust the number of codewords of each SNR through Algorithm 3.

It can be observed that N_s is the nearest integer rounded to the product of the mini-batch size N_b and the set of the SNR ratios (lines 2). As the summation of N_s should equal N_b, we may need to randomly adjust (add or delete) the value of N_s (lines 5-15).

Example: Let $a = 8$, the mini-batch size $N_b = 128$, and the SNR training set $S = \{s_k = k, \text{for } 1 \leq k \leq a\}$. Then, the sets

Algorithm 1 Semi-Adaptive SNR Algorithm

```
Input: $L_{pre}$, $L_{cur}$, $f_{optimal}$, $f_{att}$, $T_{att}$, $N_b$, $P$
Output: $P$, $T_{att}$, $N_s$
1 SemiAdaptSNR($L_{pre}$, $L_{cur}$, $f_{optimal}$, $f_{att}$, $T_{att}$, $N_b$, $P$)
2     if $L_{cur} > (f_{optimal} \times L_{pre})$ then
3         $T_{att}$ ++;
4     end
5     if $T_{att} > 0$ then
6         $P$, $N_s$ ←AdaptiveResizePerSNR($P$, $f_{att}$, $N_b$, $T_{att}$);
7     end
8     return $P$, $T_{att}$, $N_s$;
```

Algorithm 2 Adaptively Adjust SNR Ratios Set and Resize SNR Set

```
Input: $P$, $f_{att}$, $N_b$, $T_{att}$
Output: $P$, $N_s$
1  AdaptiveResizePerSNR($P$, $f_{att}$, $N_b$, $T_{att}$)
2      $i = 0$;
3      if $T_{att} > len(P)$ then
4          $T_{att} = len(P)$;
5      end
6      while $i < T_{att}$ do
7          if $f_{att} == 1$ then
8              $p_i \leftarrow 0$;
9          else
10             $p_i \leftarrow p_i \times (1 - f_{att})^{(Tatt-i)}$;
11         end
12         $i++$;
13     end
14     $P \leftarrow Normalize(P)$;
15     $N_s \leftarrow ResizeSNRSet(N_b, P)$;
16     return $P$, $N_s$;
```

P and N_s can be determined as $P = \{p_k = 1/8, \text{for } 1 \leq k \leq 8\}$ and $N_s = \{n_s(k) = 16, \text{for } 1 \leq k \leq a\}$.

In this example, we set the desired $f_{optimal} = 0.9$ and $f_{att} = 1.0$. This means that if the decoder cannot reduce 10% of the MBBCE value of the previous epoch (line 2 in Algorithm 1), the SNR adaptation will be triggered (lines 6-13 in Algorithm 2). The set P can be updated as

$$P = \{0, 0.143, 0.143, 0.143, 0.143, 0.143, 0.143, 0.143\}$$

Thus, through Algorithm 3, N_s for the next epoch can be adjusted as follows

$$N_s = \{0, 18, 18, 20, 18, 18, 18, 18\}$$

B. AUTO-ADAPTIVE SNR ALGORITHM

Furthermore, one question comes to mind, is it possible for NNs to automatically adjust the set of the number of codewords for each SNR? We review the semi-adaptive SNR algorithm. We can conclude that the SNR adaptation is achieved by varying the set of SNR ratios P, which essentially affects the change of N_s. This shows us how to let NNs learn to automatically adjust N_s. The Auto-Adaptive SNR algorithm is described in detail:

Algorithm 3 Resize the SNR Set N_s

Input: N_b, P
Output: N_s

```
ResizeSNRSet($N_b$, $P$)
    $N_s = Round(N_b \times P)$;
    $dif = abs(N_b - Sum(N_s))$;
    $idx = randint(0, \text{len}(N_b) - 1)$;
    if $N_b > Sum(N_s)$ then
        $N_b[idx] += dif$;
    else:
        while $dif > 0$ do
            if $N_b[idx] > 1$ then
                $N_b[idx] -= 1$;
                $dif -= 1$;
            end
            $idx = randint(0, \text{len}(N_b) - 1)$;
        end
    end
    return $N_s$;
```

Algorithm 4 Auto-Adaptive SNR Algorithm

Input: P
Output: P, N_s

```
AutoAdaptSNR($P$)
    $P \leftarrow Normalize(P)$;
    $N_s \leftarrow ResizeSNRSet(N_b, P)$;
    return $P$, $N_s$;
```

1) INITIALIZATION

- Set the values of a and N_b, the sets S and N_s can be determined.
- Initialize the set of the SNR ratios $P = \{p_k = n_s(k)/N_b$, for $1 \leq k \leq a\}$. It is important to change the status of P to trainable when programing the algorithm. This is the key to enable the NN to automatically adjust the P value.

2) TRAINING

Algorithm 4 presents the realization of the auto-adaptive SNR algorithm. The code is simple and easy to implement. Because the set P is set to be trainable, the NN can automatically adjust these weights in order to optimize the average MBBCE value. Similarly, the set of SNR ratios P will be normalized and used in Algorithm 3 to process the adjustment of the number of codewords for each SNR.

Example: Let $a = 8$, the mini-batch size $N_b = 128$, and the training SNR set $S = \{s_k = k$, for $1 \leq k \leq a\}$. Then, the sets P and N_s can be determined as $P = \{p_k = 1/8$, for $1 \leq k \leq 8\}$ and $N_s = \{n_s(k) = 16$, for $1 \leq k \leq a\}$. Note that the status of the set P must be set to trainable.

It can be observed that the set P will be auto-adjusted according to the average MBBCE value for each epoch. For clarity, one simple case is used for exemplification. If the set P is adjusted by the NN to $P = \{9.81e^{-03}, 7.98e^{-03}, 6.15e^{-03}, 4.91e^{-03}, 6.54e^{-03}, 1.78e^{-01}, 3.70e^{-01}, 4.16e^{-01}\}$, then the N_s for the next epoch can be adjusted by Algorithm 3 as $N_s = \{1, 1, 1, 1, 1, 23, 47, 53\}$.

TABLE 1. Training hyper parameters.

Hyperparameters	Values
Loss Function	Average MBBCE in (12)
Optimizer	RMSPROP
SNR range	1 dB to 8 dB
Learning Rate	0.001
Batch Size	128 / 256 (*)
Message Range	[-10, 10]
$f_{optimal}$	0.9 (**)
f_{att}	1.0 (**)

(*) for BCH(127, 106) code and (155, 64) LDPC code, respectively.
(**) for Semi-Adaptive SNR algorithm.

IV. RESULTS AND DISCUSSION

We present the results of training and applying the proposed algorithms to two different codes, BCH(127, 106) [24] code and (155, 64) LDPC code [25]. The decoders were built in Python 3.7 with the Tensorflow library. The training is performed by using the SGD with mini-batch, and the optimizer for training the neural network is set to RMSPROP [26]. All other training relevant hyperparameters are summarized in Table 1. Let the number of iterations T of the SPA decoder and all neural decoders be 5. Therefore, all neural decoders have 10 hidden layers. The training dataset is generated by sending all zero codewords with varying SNRs ranging from 1 dB to 8 dB. The codewords are not all zero, the testing dataset is constructed in the same way. The input value needs to be limited to the range of [-10, 10] to make the tanh calculation in (7) and (10) stable, as with the SPA decoder.

During the training stage, the decoders are trained by 30,000 batches and validated by 10,000 batches in each epoch. The early stopping function is used to monitor the average MBBCE calculated by (16) at the end of each epoch. Once the model performance measure stops improving for 5 consecutive epochs, training stops. Only the model with the best performance will be saved. Then, this model will be used to decode the non-all-zero codewords of various noise variances at the testing stage. Simulation will be stopped at a minimum of 100 frame errors or 4,000,000 simulated frames for each SNR. If the performance is low, the model is reloaded and retrained.

Figs. 4 and 5 depict the bit-error-rate for the BCH(127, 106) code and (155, 64) LDPC code with and without semi-/auto-adaptive SNR algorithms, respectively. For comparison, the performances of these decoders trained by using pure high SNR are also included. First, the decoders trained by different types of datasets are abbreviated as follows:

1. TypeU: The dataset is composed of codewords with uniform SNR (i.e., from 1 dB to 8 dB).
2. TypeH: The dataset is composed of codewords of pure high SNR with a value of 8 dB.

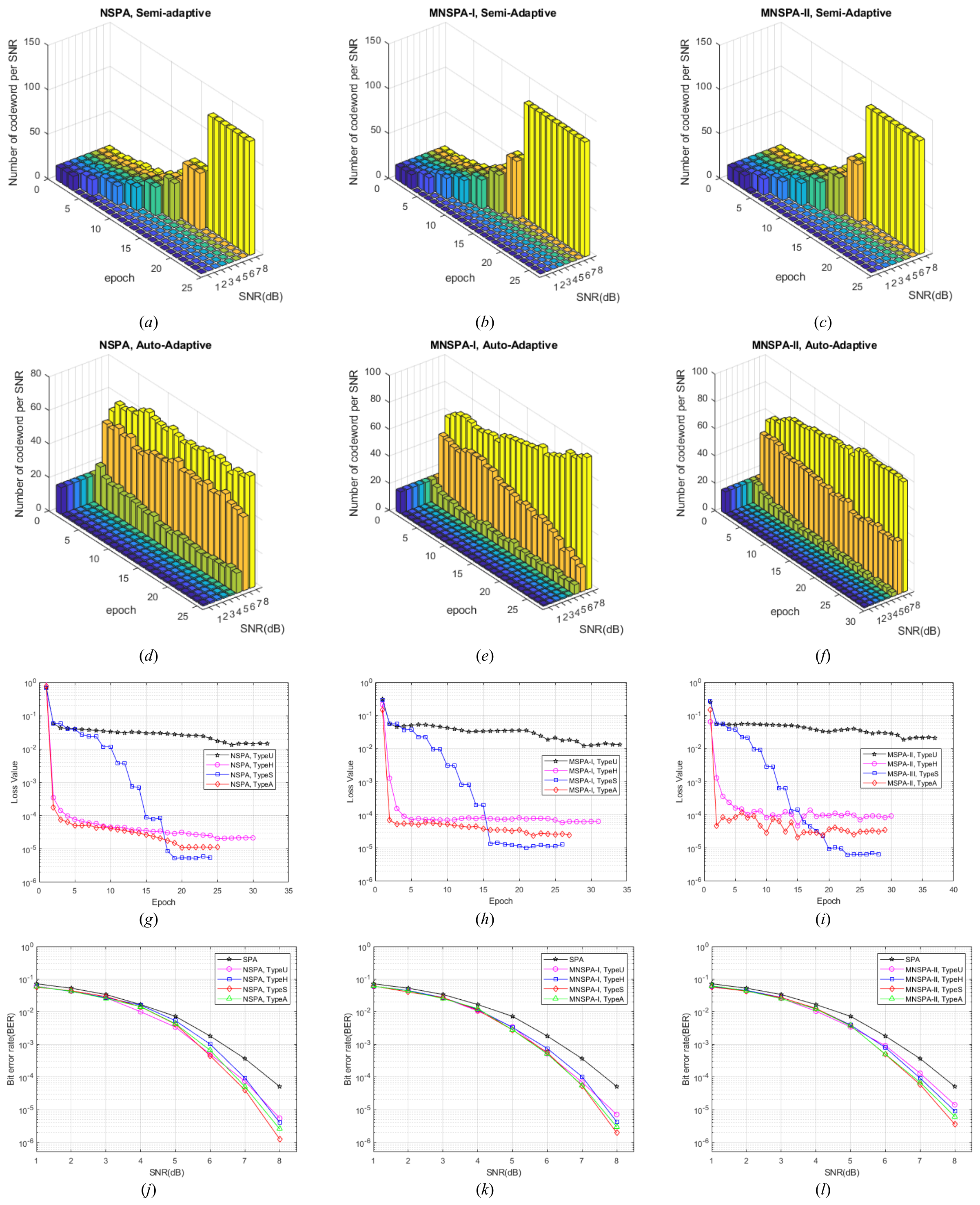

FIGURE 4. **Performance comparison of NSPA, MNSPA-I, and MNSPA-II decoders with/without semi-/auto-adaptive SNR algorithms for the BCH(127, 106) code.**

3. TypeS: The dataset is constructed using the Semi-Adaptive algorithm.
4. TypeA: The dataset is constructed using the Auto-Adaptive algorithm.

From the results in Figs. 4(a) to (c) and Figs 5(a) to (c), we can observe that the distributions of the dataset N_s for training these TypeS decoders are very similar. This is because the distribution is directly influenced by f_{att}. In this study, the value of f_{att} is set to 1, which means that if the average MBBCE value of the current epoch does not reach the desired target (i.e., $f_{optimal} \times L_{pre}$), the number of codewords with a small SNR (from 1dB to 7dB) will be set to zero in sequence.

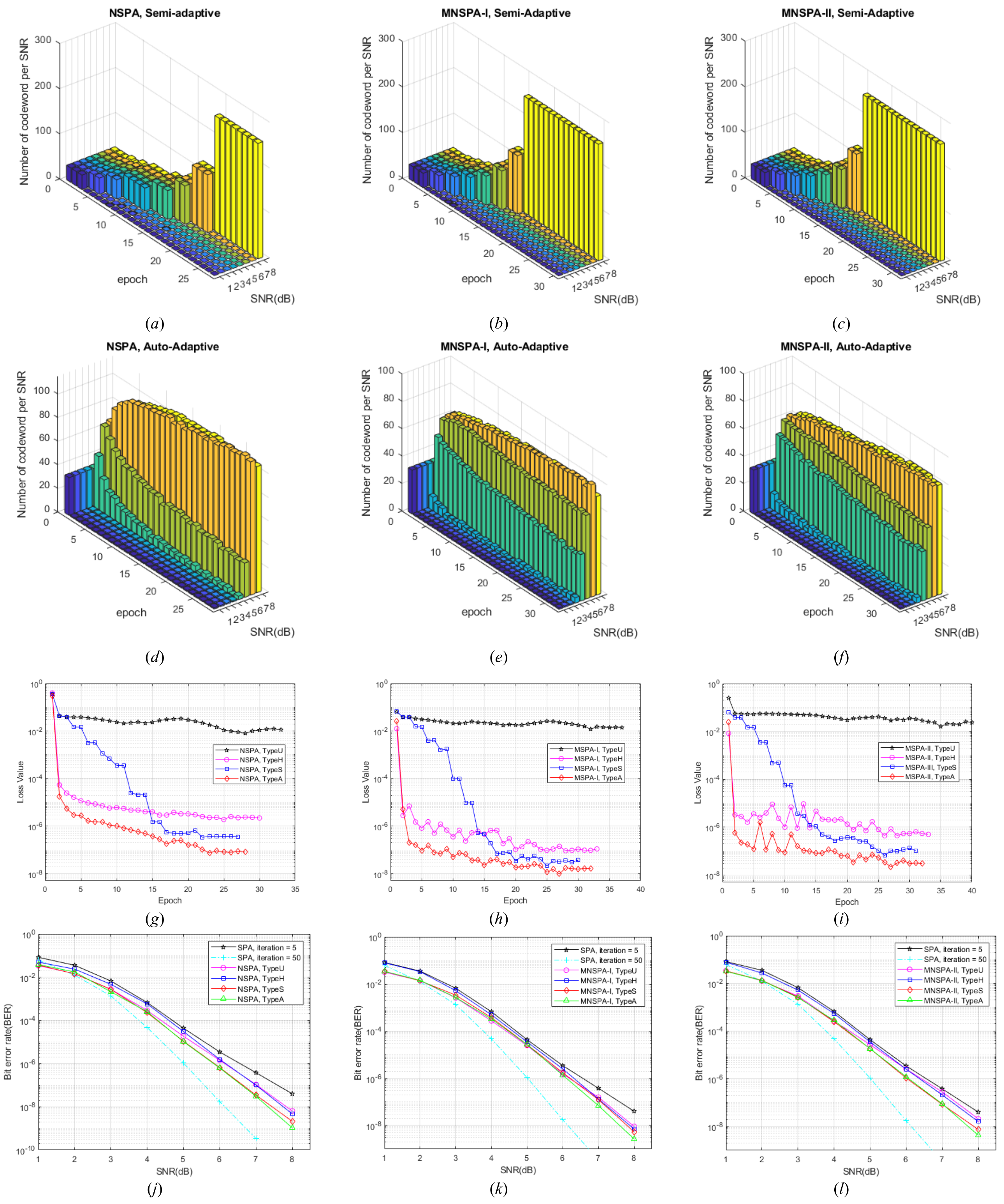

FIGURE 5. Performance comparison of NSPA, MNSPA-I, and MNSPA-II decoders with/without semi-/auto-adaptive SNR algorithms for the (155, 64) LDPC code.

Worth noting, different combinations of $f_{optimal}$ and L_{pre} generate different distributions of the dataset N_s, which may cause slightly different training results. However, this is beyond the scope of the study.

Figs. 4(d) to (f) and Figs. 5(d) to (f) present the distributions of the dataset N_s for the TypeA decoders. It can be observed that the training dataset N_s for the TypeA decoders of the BCH(127, 106) code is mainly composed of codewords

TABLE 2. Decoding results.

Code		**BCH(127, 106) code**				**(155, 64) LDPC code**			
Decoder Type	**Training Type**	TypeU	TypeH	TypeS	TypeA	TypeU	TypeH	TypeS	TypeA
NSPA	(E_{conv}, T_{ave})[1]	(27, 9038.2)	(25, 8779.2)	(19, 8822.8)	(20, 8827.8)	(28, 4210.1)	(25, 3409.2)	(22, 3448.6)	(23, 3533.1)
	Time Ratio[2]	–	89.9 %	68.7 %	72.3 %	–	72.3 %	64.4 %	68.9 %
	Gain	–	0.1 dB	0.5 dB	0.3 dB	–	0.1 dB	0.5 dB	0.6 dB
	(BER)		(5.07×10^{-6})				(6.45×10^{-9})		
MNSPA-I	(E_{conv}, T_{ave})	(29, 8141.9)	(26, 7455.9)	(21, 7595.9)	(22, 7531.1)	(32, 3619.7)	(28, 2911.6)	(25, 2943.1)	(27, 2990.3)
	Time Ratio	–	82.1 %	67.6 %	70.2 %	–	70.4 %	63.5 %	69.7 %
	Gain	–	0.2 dB	0.5 dB	0.4 dB	–	0.1 dB	0.2 dB	0. 5 dB
	(BER)[3]		(7.12×10^{-6})				(9.14×10^{-9})		
MNSPA-II	(E_{conv}, T_{ave})	(32, 7542.4)	(25, 7361.9)	(23, 7472.6)	(24, 7422.9)	(35, 3615.3)	(28, 2903.3)	(26, 2946.7)	(27, 3010.3)
	Time Ratio	–	76.3 %	71.2 %	73.8 %	–	64.2 %	60.5 %	64.2 %
	Gain	–	0.2 dB	0.6 dB	0.4 dB	–	0.1 dB	0.4 dB	0.5 dB
	(BER)		(1.41×10^{-5})				(2.09×10^{-8})		

[1] E_{conv} denotes the training epochs for these decoders to reach convergence, and T_{ave} represent the average execution time for training 30,000 batches in one epoch

with an SNR of 7 and 8 dB and partial codewords of 6 dB. For the (155, 64) LDPC code, the dataset N_s consists of a large number of codewords with an SNR of 6-8 dB and partial codewords of 5 dB.

The average MBBCE values for these two codes are presented in Figs. 4(*g*) to (*i*) and Figs. 5(*g*) to (*i*), respectively. Obviously, the MBBCE value of the TypeU decoder had the worst performance. Because the dataset consists of many codewords with a small SNR, the noise is too large to be useful for the NN to learn and recover the original codewords. The MBBCE value of the TypeS decoder appears to have sharp-edged improvements in specific epochs, where the dataset N_s has been adjusted the number of codewords of small SNR in sequence to zero. The MBBCE values of the TypeH and TypeA decoders can be improved rapidly at the second epoch. This is because the datasets for the two types of decoders are constructed by the codewords with higher SNRs, while the noise is lower when compared with the TypeU and TypeS decoders at the initial training epochs. Furthermore, we can observe that the TypeA decoder has better performance than the TypeH decoder. The main reason being that the dataset for the former consists of codewords with multi SNRs, which is more helpful for the NN to learn and optimize the weights.

From the results in Figs. 4(*j*) to (*l*), we can observe that, for the BCH(127, 106) code, the TypeS decoder outperforms the other decoders. Compared with the TypeU decoder, the improvements of the TypeS decoder and the TypeA decoder vary from 0.3 dB to 0.6 dB. For the (155, 64) LDPC code, the performance of the TypeA decoder is better than others, as shown in Figs. 5(*j*) to (*l*). Likewise, compared with the TypeU decoder, the improvements of the TypeS decoder and the TypeA decoder range from 0.2 dB to 0.6 dB. However, these neural decoders are still unable to compete with the existing excellent algorithms due to the following possible reasons.

1. **BCH code:** Since the (127,106)-BCH code has a large number of short cycles in the graph, the SPA is essentially not suitable for decoding this code. Although these neural decoders outperform the original SPA decoders, compete with the existing excellent algorithms [27], [28], they still have long way to go.
2. **(155, 64)-LDPC code:** Generally, as the number of iterations increases, higher coding gain can be obtained. As shown in Figs. 5(*j*) to (*l*), it can be observed that the SPA decoder with iteration 50 outperforms all neural decoders for at least 1.3 dB. For the neural SPA decoders, if the iteration number is set to 50, the NN will have 100 hidden layers and this deep network is hard to train because of the notorious vanishing gradient problem. To build a deeper NN, our future work will apply the residual network (ResNet) [29] to these neural networks.

The results of these decoders are summarized in Table 2. As shown in Table 2, for the BCH(127, 106) code and (155, 64) LDPC code, we can observe that the training epochs of these decoders converge in a range from 19 to 32 epochs and from 22 to 35 epochs, respectively. The average execution time T_{ave} of the former, for training 30,000 batches in one epoch, varies from 7361.9 to 9038.2 seconds, and the latter ranges from 2903.3 to 4210.1 seconds. The ratio of convergence time of the TypeS (or TypeA/TypeH) decoder to the TypeU decoder is also calculated. It can be observed that for the BCH(127, 106) code, the TypeS decoder and TypeA decoder only need 71.2% and 73.8% of the training time of the TypeU decoder at most, respectively. For the (155, 64) LDPC code, the TypeS decoder and TypeA decoders only require 64.4% and 69.7% of the training time of the TypeU decoder at most, respectively. In addition, we can observe that the TypeS decoders have the advantages in reducing the training time over the TypeA decoders. Another interesting result is that the TypeA decoders outperform the TypeS decoders for the (155, 64) LDPC code. However, for the BCH(127, 106) code, the result is just opposite. This is because that the (155, 64) LDPC code has less short cycles than the BCH(127, 106) code, the performance improvement of the former requires more training time and high-quality training data to optimize the weights of the NN. Since the TypeS decoders will sequentially attenuate the codeword number of small SNRs during the training stage, the dataset for the TypeS decoders is composed of codewords of pure SNR with a value of 8 dB in the last few training epochs. Meanwhile, the dataset for the TypeA decoders consists of codewords with multi SNRs, which is more helpful for the NN to learn and optimize the weights. Moreover, these simulation results prove that the proposed semi-/auto-adaptive SNR algorithms can not only train the neural network to achieve better performance, but also reduce the convergence time.

V. CONCLUSION

As the training data is an important factor for training a neural network, two novel semi-/auto-adaptive SNR algorithms were proposed to adjust the composition of the training dataset. The codeword number for each SNR is changed according to the optimization of the average mini-batch BCE in each epoch. For exemplification, three types of DNN decoders for the BCH(127, 106) code and (155, 64) LDPC code were trained with/without the semi-/auto-adaptive SNR algorithms, respectively. Compared with the original neural decoders, the simulation results showed that the performance of these neural decoders for the BCH(127, 106) code can be improved in the range of 0.3 to 0.6 dB and that for the (155, 64) LDPC code can be improved by 0.2 to 0.6 dB. The proposed algorithms can be easily implemented without additional decoding complexity. Moreover, the training time for these decoders to achieve convergence could be reduced by up to 28.8% for the BCH(127, 106) code and up to 35.6% for the (155, 64) LDPC code.

However, these neural decoders trained by the BCH(127, 106) code and (155, 64) LDPC code may not be sufficient to directly decode the signals under poor communication conditions (i.e., at small SNR regions), because the noise is too large for the NN to recover the original codewords. An efficient way to overcome this problem is to increase the code length of the adopted ECCs. But the main limitation is that the GPU hardware requirement increases significantly as the code length of ECCs code increases. Therefore, our future work will focus on how to integrate the ResNet into these neural decoders to build a deeper network with less weights and complexity. In addition, since the SVM-based system has the ability to predict the SNR value of the received signal with a good accuracy rate, this is helpful for the design and training of the neural decoders. We can properly split the desired SNR values into parts, and train these neural decoders with different SNR values, respectively. Then, these well-trained neural decoders can be set to the receiver end. Once the predicted SNR value is obtained, the receiver can switch to the suitable neural decoder simultaneously to recover the received signal.

REFERENCES

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, ''Near Shannon limit error-correcting coding and decoding: Turbo-codes,'' in *Proc. IEEE Int. Conf. Commun.*, Geneva, Switzerland, May 1993, pp. 1064–1070, doi: 10.1109/ICC.1993.397441.

[2] R. Gallager, ''Low-density parity-check codes,'' *IEEE Trans. Inf. Theory*, vol. IT-8, no. 1, pp. 21–28, Jan. 1962, doi: 10.1109/TIT.1962.1057683.

[3] E. Arikan, ''Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,'' *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, Jan. 2009, doi: 10.1109/TIT.2009.2021379.

[4] S. Shao, P. Hailes, T.-Y. Wang, J.-Y. Wu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, ''Survey of turbo, LDPC, and polar decoder ASIC implementations,'' *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2309–2333, 3rd Quart., 2019, doi: 10.1109/COMST.2019.2893851.

[5] Sonali, A. Dixit, and V. K. Jain, ''SNR- and rate-optimized LDPC codes for free-space optical channels,'' *IEEE Access*, vol. 9, pp. 13212–13223, 2021, doi: 10.1109/ACCESS.2021.3051687.

[6] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, ''Reduced complexity iterative decoding of low-density parity check codes based on belief propagation,'' *IEEE Trans. Commun.*, vol. 47, no. 5, pp. 673–680, May 1999, doi: 10.1109/26.768759.

[7] J. Chen and M. P. C. Fossorier, ''Density evolution for two improved BP-based decoding algorithms of LDPC codes,'' *IEEE Commun. Lett.*, vol. 6, no. 5, pp. 208–210, May 2002, doi: 10.1109/4234.1001666.

[8] M. K. Roberts, S. S. Mohanram, and N. Shanmugasundaram, ''An improved low complex offset min-sum based decoding algorithm for LDPC codes,'' *Mobile Netw. Appl.*, vol. 24, no. 6, pp. 1848–1852, Dec. 2019, doi: 10.1007/s11036-019-01392-7.

[9] T. C.-Y. Chang, P.-H. Wang, J.-J. Weng, I.-H. Lee, and Y. T. Su, ''Belief-propagation decoding of LDPC codes with variable node–centric dynamic schedules,'' *IEEE Trans. Commun.*, vol. 69, no. 8, pp. 5014–5027, Aug. 2021, doi: 10.1109/TCOMM.2021.3078776.

[10] M. K. Roberts, S. Kumari, and P. Anguraj, ''Certain investigations on recent advances in the design of decoding algorithms using low-density parity-check codes and its applications,'' *Int. J. Commun. Syst.*, vol. 34, no. 8, p. e4765, 2021, doi: 10.1002/dac.4765.

[11] T. A. Almohamad, M. F. M. Salleh, M. Mahmud, and A. H. Y. Sa'D, ''Simultaneous determination of modulation types and signal-to-noise ratios using feature-based approach,'' *IEEE Access*, vol. 6, pp. 9262–9271, 2018, doi: 10.1109/ACCESS.2018.2809448.

[12] T. A. Almohamad, M. F. M. Salleh, M. N. Mahmud, I. R. Karas, N. S. M. Shah, and S. A. Al-Gailani, ''Dual-determination of modulation types and signal-to-noise ratios using 2D-ASIQH features for next generation of wireless communication systems,'' *IEEE Access*, vol. 9, pp. 25843–25857, 2021, doi: 10.1109/ACCESS.2021.3057242.

[13] 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. (Allerton)*, Monticello, IL, USA, Sep. 2016, pp. 341–346, doi: 10.1109/ALLERTON.2016.7852251.

[14] 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 J. Sel. Topics Signal Process.*, vol. 12, no. 1, pp. 119–131, Feb. 2018, doi: 10.1109/JSTSP.2017.2788405.

[15] T. Gruber, S. Cammerer, J. Hoydis, and S. T. Brink, ''On deep learning-based channel decoding,'' in *Proc. 51st Annu. Conf. Inf. Sci. Syst. (CISS)*, Baltimore, MD, USA, Mar. 2017, pp. 1–6, doi: 10.1109/CISS.2017.7926071.

[16] H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, and P. Viswanath, ''Communication algorithms via deep learning,'' May 2018, *arXiv:1805.09317*.

[17] L. Lugosch and W. J. Gross, ''Neural offset min-sum decoding,'' in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Germany, Jun. 2017, pp. 1361–1365, doi: 10.1109/ISIT.2017.8006751.

[18] Q. Wang, S. Wang, H. Fang, L. Chen, L. Chen, and Y. Guo, ''A model-driven deep learning method for normalized min-sum LDPC decoding,'' in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, Dublin, Ireland, Jun. 2020, pp. 1–6, doi: 10.1109/ICCWorkshops49005.2020.9145237.

[19] B. Vasić, X. Xiao, and S. Lin, "Learning to decode LDPC codes with finite-alphabet message passing," in *Proc. Inf. Theory Appl. Workshop (ITA)*, San Diego, CA, USA, Feb. 2018, pp. 1–9, doi: 10.1109/ITA.2018.8503199.

[20] L. Chu, H. He, L. Pei, and R. C. Qiu, "NOLD: A neural-network optimized low-resolution decoder for LDPC codes," *J. Commun. Netw.*, vol. 23, no. 3, pp. 159–170, Jun. 2021, doi: 10.23919/JCN.2021.000014.

[21] C.-M. Huang, C.-C. Yang, E. Wijanto, and H.-C. Cheng, "Analysis of deep multilayer perceptron neural network in MWC coded optical CDMA system with LDPC code," *Opt. Fiber Technol.*, vol. 60, Dec. 2020, Art. no. 102385, doi: 10.1016/j.yofte.2020.102385.

[22] M. Lian, F. Carpi, C. Hager, and H. D. Pfister, "Learned belief-propagation decoding with simple scaling and SNR adaptation," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Paris, France, Jul. 2019, pp. 161–165, doi: 10.1109/ISIT.2019.8849419.

[23] I. Be'Ery, N. Raviv, T. Raviv, and Y. Be'Ery, "Active deep decoding of linear codes," *IEEE Trans. Commun.*, vol. 68, no. 2, pp. 728–736, Feb. 2020, doi: 10.1109/tcomm.2019.2955724.

[24] M. Helmling. (2019). *Database of Channel Codes and ML Simulation Results*. [Online]. Available: https://www.uni-kl.de/channelcodes

[25] J.-F. Huang, C.-M. Huang, and C.-C. Yang, "Construction of one-coincidence sequence quasi-cyclic LDPC codes of large girth," *IEEE Trans. Inf. Theory*, vol. 58, no. 3, pp. 1825–1836, Mar. 2012, doi: 10.1109/TIT.2011.2173246.

[26] T. Tieleman and G. Hinton, "Lecture 6.5-RMSPROP: Divide the gradient by a running average of its recent magnitude," *COURSERA, Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.

[27] M. P. C. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Trans. Inf. Theory*, vol. 41, no. 5, pp. 1379–1396, Sep. 1995, doi: 10.1109/18.412683.

[28] S. Fragiacomo, C. Matrakidis, and J. O'Reilly, "Novel near maximum likelihood soft decision decoding algorithm for linear block codes," *IEE Proc.-Commun.*, vol. 146, no. 5, pp. 265–270, Oct. 1999, doi: 10.1049/ip-com:19990595.

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

CHUN-MING HUANG received the B.S., M.S., and Ph.D. degrees from the Department of Electrical Engineering, National Cheng Kung University, Taiwan, in 2000, 2005, and 2009, respectively. From 2010 to 2018, he was working with the National Chung-Shan Institute of Science and Technology, Longtan, Taiwan, as an Assistant Scientist. Since 2018, he has been with the Faculty of National Formosa University, Yunlin, Taiwan, where he is currently an Assistant Professor with the Department of Electronic Engineering. His research interests include error control codes and optical communications.