Deep Learning Based Modified Message Passing Algorithm for Sparse Code Multiple Access

Lanping Li*, Xiaohu Tang* and Chintha Tellambura†
* Laboratory of Information Coding and Transmission, Southwest Jiaotong University,
Chengdu 611756, China, E-mail: lanping@my.swjtu.edu.cn, xhutang@swjtu.edu.cn
† Department of Electrical and Computer Engineering,University of Alberta,
Edmonton T6G 2V4, Canada, E-mail:ct4@ualberta.ca

***Abstract*—Shuffled message passing algorithm (SMPA) is a serial variant of message passing algorithm (MPA) for sparse code multiple access (SCMA) signal detection, which accelerates the convergence rate of MPA. However, SMPA still achieves the near-optimal performance due to the effect of cycles in the factor graph. In the paper, we propose to optimize the weights assigned to the edges of the factor graph by unfolding SMPA as layers of deep neural network (DNN). We consider the weights as network parameters and then train the network offline to obtain weights which can minimize the loss function. With simulations, we show that DNN based SMPA (DNN-SMPA) outperforms SMPA in terms of bit-error-rate (BER) for the same level of computational complexity.**

***Index Terms*—sparse code multiple access (SCMA), shuffled, message passing algorithm (MPA), deep learning (DL).**

I. Introduction

Sparse code multiple access (SCMA) [1], [2] is a potential technology to reach ultra-high traffic rates, high spectral efficiency, massive connectivity and low latency of fifth generation (5G) wireless networks. To achieve these advances, low-complexity signal detection can be performed in SCMA with the message passing algorithm (MPA) based on a factor graph. Moreover, to reduce the computational complexity of MPA, [3] proposed shuffled MPA (SMPA) which accelerates the convergence rate of MPA by propagating the message immediately in current iteration. However, due to the existence of cycles in the factor graph, MPA and SMPA do not attain the optimal maximum likelihood (ML) detection performance.

Recently, to improve SCMA, deep learning techniques have been used. [4] constructs codebooks and decodes the SCMA codewords by autoencoder. But the decoder in autoencoder is of a full connectivity layer to approximate the ML function, which causes the increase of training and decoding times since the structure of traditional detection algorithm is not taken into account. To overcome this, similar to unfolding belief-propagation of channel decoding for Bose-Chaudhuri-Hocquenghem (BCH) code [5] and polar code [6], recently, [7] proposed to unfold MPA to a deep neural network (DNN) and optimize the weights added to the edges of the factor graph to reduce the effect of cycle. However, this DNN based MPA (DNN-MPA) may exhibit slow convergence inherited from unrolling MPA.

978-1-7281-1669-3/19/$31.00 ©2019 IEEE

In this paper, we therefore propose DNN-SMPA which exploits SMPA to improve the convergence rate while uses DNN to reduce the effects of factor-graph cycles. Particularly we assign weights to the edges of the factor graph and unfold the SMPA to form the desired DNN. The weights as DNN parameters can be learned by minimizing the error between the estimated symbols and transmitted symbols, i.e., the loss function. Once the training is completed, the learned weights can be adopted in SMPA to reduce the effects of cycles in the factor graph. The simulation results show that the DNN-SMPA improves bit-error-rate (BER) over traditional SMPA for the same number of iterations, especially at the high E_b/N_0 regime.

II. System Model

A. SCMA Model

The SCMA system has J users (layers) spreading over K shared orthogonal resource blocks (e.g., orthogonal frequency division multiplexing (OFDM) sub-carriers or multiple input multiple output (MIMO) spatial streams). For $j = 1, \ldots, J$, $\boldsymbol{b}_j$ is the bit streams of user j, which is mapped into K-dimensional complex codeword $\boldsymbol{x}_j$ from the predefined codebook of size M_j. SCMA selects sparse K-dimensional complex codewords (i.e., only $N_j \leq K$ non-zero elements exist). We assume that all users have the same constellation size and the same number of non-zero elements, i.e., $M_j = M, N_j = d_v, j = 1, 2, \cdots, J$.

The uplink received signal can be written as

$$\boldsymbol{y} = \sum_{j=1}^{J} \mathrm{diag}(\boldsymbol{h}_j)\boldsymbol{x}_j + \boldsymbol{z}, \tag{1}$$

where $\boldsymbol{h}_j = (h_{j,1}, h_{j,2}, \cdots, h_{j,K})^T$ is the channel gain vector of user j, $\boldsymbol{x}_j = (x_{j,1}, x_{j,2}, \cdots, x_{j,K})^T$ is the K-dimensional codeword of user j, and $\boldsymbol{z}$ is a complex additive white Gaussian noise with zero mean and $\sigma^2\boldsymbol{I}_K$ variance.

In fact, this system can be depicted by a $K \times J$ indicator matrix $\boldsymbol{F}$, where K resources are occupied by J users. Especially, the number of users interfering at each resource node is d_c (row weight of $\boldsymbol{F}$), ξ_k is the set of users collided over resource k, the spreading resource of each user is d_v

(column weight of $\boldsymbol{F}$), ζ_j is the set of resources occupied by user j. For example, indicator matrix

$$\boldsymbol{F} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{2}$$

is for $K = 4$, $J = 6$, $d_c = 3$, $d_v = 2$. $\xi_1 = \{1, 3, 5\}$, $\zeta_1 = \{1, 3\}$, and so on, whose factor graph is illustrated in Fig. 1.

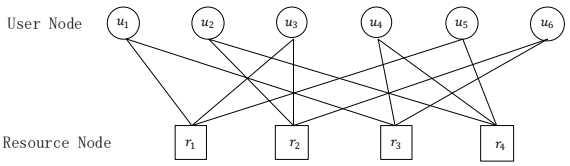

Fig. 1: Factor graph of SCMA with $K = 4$, $J = 6$.

B. MPA and DNN-MPA for SCMA

MPA based on the SCMA factor graph was presented in [8]. Indeed, MPA exchanges the extrinsic information between user nodes and resource nodes along the edges iteratively.

At the beginning, all symbols have equal probability (no prior information). So the messages $I^0_{u_j \to r_k}(x_j)$ ($j = 1, \cdots, J$, $k = 1, \cdots, K$) are initialized to 0 in the log domain.

For the i-th ($i = 1, \ldots, I$) iteration, the message sent from k-th ($k = 1, \ldots, K$) resource node to j-th ($j \in \xi_k$) user node, i.e., resource node message, is updated by using

$$\begin{aligned} & I^i_{r_k \to u_j}(x_j) \\ = \; & \ln\Big(\sum_{\tilde{\boldsymbol{x}}: \tilde{x}_j = x_j} \exp\big(M_k(\tilde{\boldsymbol{x}}) + \sum_{\tilde{j} \in \xi_k \setminus \{j\}} I^{i-1}_{u_{\tilde{j}} \to r_k}(\tilde{x}_{\tilde{j}})\big)\Big) \\ \approx \; & \max_{\tilde{\boldsymbol{x}}: \tilde{x}_j = x} \big(M_k(\tilde{\boldsymbol{x}}) + \sum_{\tilde{j} \in \xi_k \setminus \{j\}} I^{i-1}_{u_{\tilde{j}} \to r_k}(\tilde{x}_{\tilde{j}})\big), \end{aligned} \tag{3}$$

where $M_k(\tilde{\boldsymbol{x}}) = \frac{-\|\boldsymbol{y}_k - \sum_{p \in \xi_k} h_{p,k} \tilde{x}_{p,k}\|^2}{2\sigma^2}$, and the message sent from j-th ($j = 1, \ldots, J$) user node to k-th ($k \in \zeta_j$) resource node for i-th iteration, i.e., user node message, is updated by

$$I^i_{u_j \to r_k}(x_j) = \sum_{\tilde{k} \in \zeta_j \setminus \{k\}} I^i_{r_{\tilde{k}} \to u_j}(x_j). \tag{4}$$

After I iterations, the symbol probability of user j ($j = 1, \ldots, J$) is decided by

$$I_{u_j}(x_j) = \sum_{k \in \zeta_j} I^I_{r_k \to u_j}(x_j). \tag{5}$$

Since MPA is based on the factor graph, it is not optimal when there exists cycles in the graph. To eliminate the impact of cycles, [7] proposed DNN-MPA by unfolding MPA where multiplicative weights were assigned. Note that the computational complexity of the MPA algorithm is mainly determined by the update of the resource node message. In fact, the weights of $M_k(\tilde{\boldsymbol{x}})$ and $I^{i-1}_{u_{\tilde{j}} \to r_k}(\tilde{x}_{\tilde{j}})$ in (3) not only increase the computational complexity, but also increase the number of training variables.

Thus, in this paper, weights in DNN-MPA are assigned only to the user node message and the decision message, i.e., (4) and (5) are respectively replaced by

$$I^i_{u_j \to r_k}(x_j) = \sum_{\tilde{k} \in \zeta_j \setminus \{k\}} w^i_{\tilde{k},j} I^i_{r_{\tilde{k}} \to u_j}(x_j), \tag{6}$$

and

$$I_{u_j}(x_j) = \sum_{k \in \zeta_j} w_{out,k,j} I^I_{r_k \to u_j}(x_j), \tag{7}$$

where $w^i_{k,j}$ and $w_{out,k,j}$ are the multiplicative weights.

Unfortunately, DNN-MPA also retains the disadvantage of slow convergence because it is based on unfolding MPA.

C. SMPA for SCMA

Although the original MPA can approximate the ML efficiently, it is not efficient in terms of computational complexity. From (3), we can see that the original MPA only utilizes the user node messages obtained at $(i-1)$-th iteration to update resource node at the i-th iteration. To accelerate the convergence rate of MPA, [3] proposed SMPA, which enables immediate propagation of updated message in the current iteration, i.e., (3) can be replaced by

$$\begin{aligned} I^i_{r_k \to u_j}(x_j) \; = \; & \max_{\tilde{\boldsymbol{x}}: \tilde{x}_j = x_j} \big(M_k(\tilde{\boldsymbol{x}}) + \sum_{\hat{j} \in \xi_k \setminus \{j\}, \hat{j} < j} I^i_{u_{\hat{j}} \to r_k}(\tilde{x}_{\hat{j}}) \\ & + \sum_{\tilde{j} \in \xi_k \setminus \{j\}, \tilde{j} > j} I^{i-1}_{u_{\tilde{j}} \to r_k}(\tilde{x}_{\tilde{j}})\big). \end{aligned} \tag{8}$$

III. SMPA Based on Deep Learning

In this section, to overcome the disadvantage of slow convergence in DNN-MPA, we utilize the structure of SMPA with deep learning. Specifically, we assign the multiplicative weights to the edges of factor graph, which are trained offline by unrolling the SMPA instead of MPA to a DNN. After training, the weights are used in online deployment of SMPA. This is the key idea of our proposed DNN-SMPA.

A. Structure of DNN-SMPA

In DNN-SMPA network, there are $T = 2JI + 3$ layers containing input layer, $2JI+1$ hidden layers and output layer. The input layer is fed into initialization of prior message $I^0_{u_j \to r_k}(x_j) = 0$, received signal $\boldsymbol{y}$ and codebooks.

Among the hidden layers, $2JI$ layers can be constructed by unfolding SMPA with I iterations and $2J$ layers for each iteration, in which message of resource node and message of user node are sequentially updated from user 1 to user J according to (8) and (6) respectively.

Generally, consider t-th hidden layer ($t = 1, 2, \cdots, 2JI$) as shown in Fig. 2, for odd values $t = 2J(i-1) + 2j - 1$, it represents the update of resource node message of user $j = \frac{t+1-2J(i-1)}{2}$ at the ($i = \lfloor \frac{t-1}{2J} \rfloor + 1$)-th iteration, where $\lfloor . \rfloor$ denotes floor function. The outputs are $I^i_{r_k \to u_j}$ generated by (8) with $k \in \zeta_j = \{k_1, k_2\}$. The inputs are received signal

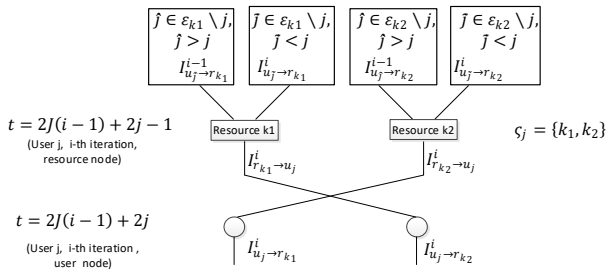

Fig. 2: Every two hidden layers of DNN-SMPA network by unfolding message of user j at i-th iteration.

$\boldsymbol{y}$, codebooks, the i-th iteration user node message $I^{i}_{u_{\hat{j}} \to r_k}$ and the $(i-1)$-th iteration user node message $I^{i-1}_{u_{\tilde{j}} \to r_k}$ with $\hat{j} \in \xi_k \setminus j, \hat{j} < j$ and $\tilde{j} \in \xi_k \setminus j, \tilde{j} > j$ for resource node k. Each input and output variable are of size M.

For even values $t = 2J(i-1)+2j$, it represents the update of user node message of user $j = \frac{t-2J(i-1)}{2}$ at the $(i = \lfloor \frac{t}{2J} \rfloor + 1)$-th iteration. The outputs are $I^{i}_{u_j \to r_k}$ yielded by (6) with $k \in \zeta_j = \{k_1, k_2\}$. The inputs are i-th iteration resource node message $I^{i}_{r_k \to u_j}$.

At the last (the $t = 2JI + 1$-th) hidden layer, it represents decision message produced by (7) with resource node message of the $(\lfloor \frac{t-1}{2J} \rfloor)$-th iteration as input. The variables $\boldsymbol{\theta} = \left\{ w^{i}_{\tilde{k},j}, w_{out,k,j} \right\}$ in (6) and (7) are weight parameters assigned to factor graph, which should be trained to reduce the effects of the loops. The structures of different schemes are listed in Table I.

TABLE I: comparison of different schemes

Schemes	MPA	DNN-MPA	SMPA	DNN-SMPA
resource node message	(3)	(3)	(8)	(8)
user node message	(4)	(6)	(4)	(6)
decision message	(5)	(7)	(5)	(7)

The output layer of DNN-SMPA network is to transfer the decision message (7) from the likelihood domain to the probability domain. Thus, the softmax function is used in the last hidden layer for each user $\boldsymbol{z}_j = I_{u_j}(x)$ as follows

$$\sigma(\boldsymbol{z}_j)_m = \frac{e^{z_{j,m}}}{\sum_{m=1}^{M} e^{z_{j,m}}}, \quad m = 1, \cdots, M. \tag{9}$$

Let $\hat{\boldsymbol{s}}_j = [\sigma(\boldsymbol{z}_j)_1, \cdots, \sigma(\boldsymbol{z}_j)_M]$ be the estimated symbols of user j. Then, the final output of network is $\hat{\boldsymbol{s}} = [\hat{\boldsymbol{s}}_1, \cdots, \hat{\boldsymbol{s}}_J]$.

B. Training of DNN-SMPA

Parameters $\boldsymbol{\theta}$ in DNN-SMPA network can be optimized to reduce the effect of loop. We initialize $\boldsymbol{\theta}$ with all ones since traditional SMPA factor graph has unit weights in all the branches.

To train DNN-SMPA network, we need generate labeled training samples, i.e., a set of input-output vector pairs $(\boldsymbol{s}^d, \hat{\boldsymbol{s}}^d)$, $d = 1, \cdots, D$, where $\hat{\boldsymbol{s}}^d$ is the desired output of the DNN-SMPA network for the transmitted symbols $\boldsymbol{s}^d = [\boldsymbol{s}^d_1, \cdots, \boldsymbol{s}^d_J]$. Note that $\boldsymbol{s}^d_j$ is the one-hot vector of bit stream $\boldsymbol{b}_j$ of user j's d-th sample. One-hot vector is an M-dimensional vector, the m-th element of which is equal to one and zero otherwise. For instance, $[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]$ are respectively the one-hot vectors of $\boldsymbol{b}_j = [0,0], [0,1], [1,0], [1,1]$. To generate d-th training sample, $J\log_2(M)$ bits are randomly chosen and every $\log_2(M)$ bits corresponding $\boldsymbol{b}_j$ transfer to one-hot vector $\boldsymbol{s}^d_j$ as label data. Bit stream $\boldsymbol{b}_j$ is mapped into complex codeword $\boldsymbol{x}_j$. After $\boldsymbol{x}_j$ passing through the channel, we get the received signal. Upon reception of $\boldsymbol{y}$, with codebooks and initialization prior message $I^0_{u_j \to r_k}(x_j) = 0$, the receiver applies the DNN-SMPA network to obtain the estimation $\hat{\boldsymbol{s}}^d$ of the transmitted message $\boldsymbol{s}^d$.

Algorithm 1 Training of DNN-SMPA

1) Initialize: Set $\boldsymbol{\theta} = \mathbf{1}$, i.e., $w^{i}_{\tilde{k},j} = 1$ and $w_{out,k,j} = 1$ for $k \in \zeta_j$, $\tilde{k} \in \zeta_j \setminus \{k\}$, $j = 1, \cdots, J$, $i = 1, \cdots, I$.
2) For $N_b = 1, \cdots, batches$
   a) Generating training samples: Set data set $\boldsymbol{G} = \emptyset$.
   For $d = 1, \cdots, D$
      i) Randomly generate bit stream $\boldsymbol{b}_j$ for each user j; Transfer $\boldsymbol{b}_j$ to one-hot vector $\boldsymbol{s}^d_j$;
      ii) Mapping $\boldsymbol{b}_j$ into complex codewords $\boldsymbol{x}_j$; After channel, get received signal $\boldsymbol{y}$ by (1);
      iii) Feed $\boldsymbol{y}$, codebooks and $I^0_{u_j \to r_k}(x_j) = 0$ into the DNN-SMPA network under current parameters $\boldsymbol{\theta}$, get the output $\hat{\boldsymbol{s}}^d_j$ for each user j ;
      iv) Store $(\boldsymbol{s}^d, \hat{\boldsymbol{s}}^d)$ in $\boldsymbol{G}$, where $\boldsymbol{s}^d = [\boldsymbol{s}^d_1, \cdots, \boldsymbol{s}^d_J]$ and $\hat{\boldsymbol{s}}^d = [\hat{\boldsymbol{s}}^d_1, \cdots, \hat{\boldsymbol{s}}^d_J]$.
   End for
   b) Updating parameters:
   Compute loss function by (10) ;
   Update $\boldsymbol{\theta} = \left\{ w^{i}_{\tilde{k},j}, w_{out,k,j} \right\}$ by RMSProp (11).
   End for
3) Return $\boldsymbol{\theta} = \left\{ w^{i}_{\tilde{k},j}, w_{out,k,j} \right\}$

The goal of the training process is to minimize the loss function which is the difference between the output of the network and the transmitted data. In our experiment settings, the loss function is

$$L(\boldsymbol{\theta}) = \frac{1}{D} \sum_{d=1}^{D} \|\boldsymbol{s}^d - \hat{\boldsymbol{s}}^d\|^2, \tag{10}$$

where $\boldsymbol{\theta} = \left\{ w^{i}_{\tilde{k},j}, w_{out,k,j} \right\}$ are parameters to be learned.

One popular algorithm to find $\boldsymbol{\theta}$ is root mean square prop (RMSProp) which starts with initial values $\boldsymbol{\theta} = \mathbf{1}$ in the paper and then updates $\boldsymbol{\theta}$ iteratively as

$$\begin{cases} \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{1}{\sqrt{\boldsymbol{n}} + \varepsilon} \odot \nabla L(\boldsymbol{\theta}) \\ \boldsymbol{n} \leftarrow \nu \boldsymbol{n} + (1-\nu) \nabla L(\boldsymbol{\theta}) \odot \nabla L(\boldsymbol{\theta}) \end{cases}, \tag{11}$$

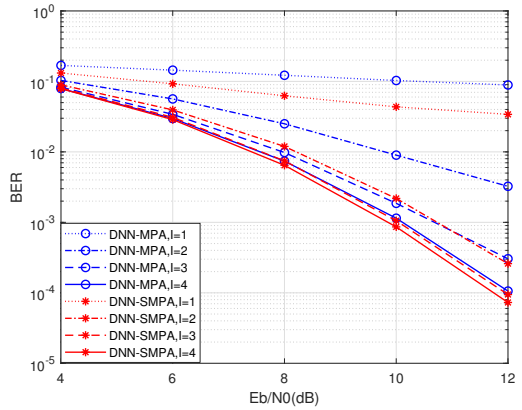

Fig. 3: BER of DNN-SMPA for one to four iterations.

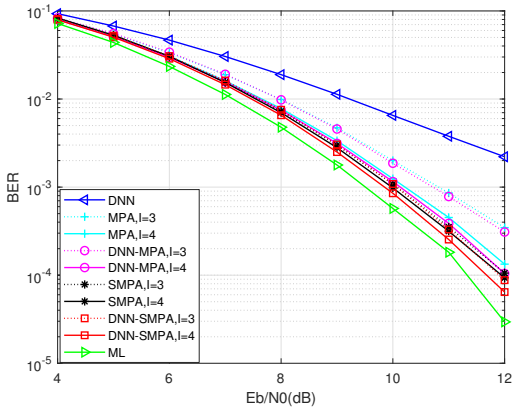

Fig. 4: BER of different algorithms in AWGN.

where $\odot$ is element-wise product, ∇ is gradient operation, the learning rate α is a hypeparameter, and the parameters $\nu = 0.9$ and $\varepsilon = 10^{-10}$ are set by default.

After several batches of generating training samples and updating parameters, we get $\boldsymbol{\theta}$. The process of training DNN-SMPA is shown in Algorithm 1.

IV. Simulation Results

In this section, we evaluate the BERs of DNN-SMPA, DNN, MPA, DNN-MPA, SMPA and ML in AWGN and Rayleigh-fading channels. The factor graph of SCMA system is determined by the matrix in (2), i.e., $K = 4$, $J = 6$. The codebooks in [9] is used for $M = 4$.

To train the DNN-SMPA, we build the network on the deep learning framework Tensorflow [10]. Training is conducted by RMSProp with the learning rate of 0.0001. 40000 batches of training dataset are used and the number of training samples in each batch is set to $D = 400$. The parameters such as learning rate, D in deep learning are brute-force searched.

As a point of comparison, DNN-MPA and DNN are considered as well. In the paper, DNN-MPA is built by (3), (6), (7), (9) and DNN is constructed by fully connected layer and rectifier linear unit (ReLU). We set up the DNN network consisting of eight layers, six of which are hidden layers. The numbers of neurons in each layer are $2K$,100,100,100,100,100,100,MJ, respectively. For a fair comparison, the same training parameters are adopted in DNN-MPA, DNN and DNN-SMPA.

A. Convergence

Fig. 3 shows the effect of the number of iterations to BER of DNN-SMPA and DNN-MPA. Thus, DNN-SMPA exhibits faster convergence than DNN-MPA.

B. BER

Fig. 4 presents the raw BERs of DNN-SMPA, DNN, MPA, DNN-MPA, SMPA and ML for AWGN channels in $I = 3$ and $I = 4$ iterations. During training stage, the training samples in DNN-SMPA, DNN-MPA and DNN are all generated under multiple E_b/N_0: $6, 8, 10, 12$ dB. We can see that at low

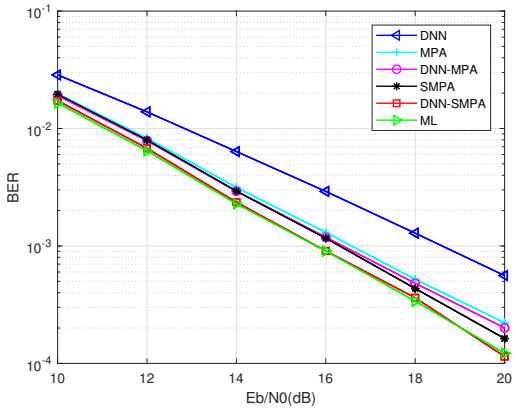

Fig. 5: BER of all the algorithms in Rayleigh channels.

E_b/N_0, DNN-SMPA has almost the same BER as SMPA. At higher E_b/N_0, the DNN-SMPA outperforms SMPA about 0.3 dB in 4 iterations. We also can see that the DNN-SMPA and SMPA in 3 iterations outperform the DNN-MPA and MPA in 4 iterations. ML performs the best but has the highest computational complexity. Whereas, DNN performs the worst because it learns slowly due to not exploiting relationships of $\boldsymbol{x}_j$ and $\boldsymbol{y}$ in (1). It can be improved by increasing the number of layers or neurons or training samples.

Fig. 5 compares BERs of DNN-SMPA, DNN, MPA, DNN-MPA, SMPA and ML in $I = 3$ iterations, where the channel experiences Rayleigh fading. It is seen that DNN-SMPA achieves nearly the same BER as ML and has about 0.4 dB, 0.8 dB, 1.0 dB gain over SMPA, DNN-MPA and MPA, respectively. While, DNN still has the worst performance.

Fig. 6 depicts the BER of SCMA for $I = 2$ iterations with two polar codes under AWGN channels. They are of code rate 1/2 and denoted by $C_1 = (16, 8)$ and $C_2 = (64, 32)$ of codeword length $N = 16$ and $N = 64$, respectively. Polar code of block size $N = 2^n$ is obtained by $\boldsymbol{c} = \boldsymbol{u}\boldsymbol{G}_N$ [11], where $\boldsymbol{u}$ contains k information bits and $N - k$ frozen positions. Generator matrix is defined by $\boldsymbol{G}_N = \boldsymbol{B}_N \boldsymbol{F}_2^{\otimes n}$, where $\boldsymbol{B}_N$

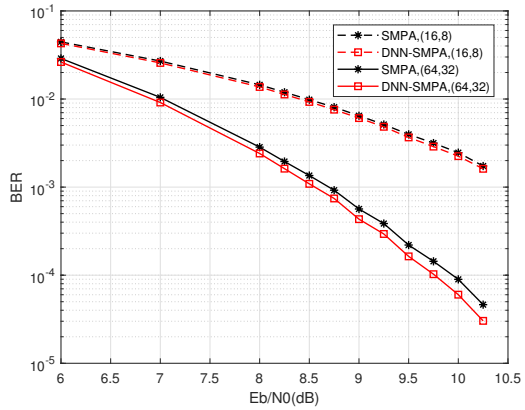

Fig. 6: BER with polar codes in AWGN

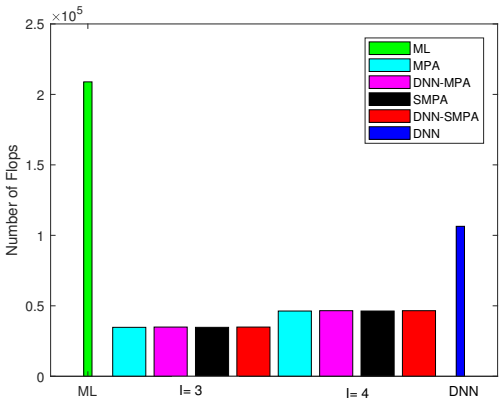

Fig. 7: Computational complexity of different schemes.

is a permutation matrix and $\boldsymbol{F}_2^{\otimes n}$ is n-th Kronecker power of $\boldsymbol{F}_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$. Belief Propagation (BP) decoding with 10 iterations is used for polar decoding. We observe that the larger the code length, the greater the performance gain of DNN-SMPA over SMPA, i.e., the gain is 0.08 dB and 0.25 dB for C_1 and C_2, respectively.

C. Complexity

In this subsection, we analyze computational complexity and storage complexity of different methods.

For ML, the receiver exhaustively searches all M^J possible combinations of codewords from users. It requires $2K(d_c - 1) + 2K + K - 1 = 2Kd_c + K - 1$ real domain additions and $6K$ real domain multiplications in every combination, so the total computational complexity of ML is $M^J(2Kd_c + 7K - 1)$ real floating-point arithmetic operations (flops). Given the same number of iterations I, the computational complexity of SMPA is the same as that of the original MPA, which is $I(Kd_cM^{d_c}(3d_c+6)+Kd_cM)+IJd_vM(d_v-2)+JM(d_v-1)$ flops [8]. Due to the multiplication of the weighting factor in (6) and (7), DNN-SMPA requires more $(I+1)Jd_vM$ flops than SMPA, which is negligible. Thus, DNN-SMPA, SMPA, DNN-MPA and MPA have roughly the same complexity. In DNN network, the computational complexity is $2N_iN_o$ in each layer where it contains N_i input neurons and N_o output neurons. Thus, the computational complexity of DNN-SMPA with $I = 4$ is lower than that of DNN network, which is $106400 = 2(2K \times 100 + 5 * 100 \times 100 + 100 \times MJ)$. The comparison results of computational complexity of different schemes are illustrated in Fig. 7 .

We know that the less training variables in deep learning, the less storage space. Consider the number of trainable variable, there are only $IJd_v + Jd_v$ trainable parameters in the DNN-SMPA and DNN-MPA network, since each iteration contains Jd_v adjustable variables following (6) and decision message also has Jd_v variables following (7). While there are $N_iN_o + N_o$ trainable variables in each layer of the DNN network. So the training variables of DNN-SMPA is less than that of DNN.

V. Conclusions

In this paper, we developed a new DNN-SMPA detector. It outperforms SMPA, DNN-MPA and MPA given the same number of iterations. By optimizing the weights assigned to the edges of the factor graph, it is shown that DNN-SMPA can reduce the effect of cycles.

References

[1] H. Nikopour and H. Baligh, "Sparse code multiple access," in *Proc. IEEE 24th Int. Symp. Pers. Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2013, pp. 332–336.

[2] L. Li, J. Wen, X. Tang, and C. Tellambura, "Modified sphere decoding for sparse code multiple access," *IEEE Commun. Lett.*, vol. 22, no. 8, pp. 1544–1547, Aug. 2018.

[3] Y. Du, B. Dong, Z. Chen, J. Fang, and L. Yang, "Shuffled multiuser detection schemes for uplink sparse code multiple access systems," *IEEE Commun. Lett.*, vol. 20, no. 6, pp. 1231–1234, Jun. 2016.

[4] M. Kim, N.-I. Kim, W. Lee, and D.-H. Cho, "Deep learning aided SCMA," *IEEE Commun. Lett.*, vol. 22, no. 4, pp. 720–723, Apr. 2018.

[5] L. Lugosch and W. J. Gross, "Neural offset min-sum decoding," in *Proc. Int. Symp. Inform. Theory (ISIT)*, Jun. 2017, pp. 1361–1365.

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

[7] C. Lu, W. Xu, H. Shen, H. Zhang, and X. You, "An enhanced SCMA detector enabled by deep neural network," in *Proc. Int. Conf. Commun. China (ICCC)*, Aug. 2018, pp. 835–839.

[8] F. Wei and W. Chen, "Low complexity iterative receiver design for sparse code multiple access," *IEEE Trans. Commun.*, vol. 65, no. 2, pp. 621–634, Feb. 2017.

[9] S. Zhang, K. Xiao, B. Xiao, Z. Chen, B. Xia, D. Chen, and S. Ma, "A capacity-based codebook design method for sparse code multiple access systems," in *Proc. Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Oct. 2016, pp. 1–5.

[10] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, vol. 16, Nov. 2016, pp. 265–283.

[11] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inform. Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.