International Journal of Innovative
Computing, Information and Control
Volume **17**, Number **3**, June **2021**

ICIC International ©2021 ISSN 1349-4198
pp. 789–805

A RECURRENT NEURAL NETWORK-BASED SUCCESSION CANCELLATION FOR POLAR DECODER

Guiping Li, Xiuhua Hu and Junjun Guo

School of Computer Science and Engineering
Xi'an Technological University
No. 2, Xuefuzhonglu Road, Weiyang District, Xi'an 710021, P. R. China
15693685@qq.com; llxghp@sina.com; 15332400693@189.com

Received November 2020; revised March 2021

Abstract. *To solve the low output and high latency of successive cancellation decoding for polar codes in 5G scene, accelerate scheme of successive cancellation (SC) decoding based on recurrent neural network (RNN) for polar codes is proposed. In this work, we leverage the expert knowledge in communication systems and adopt a long short-term memory network (LSTM)-aided SC to improve the performance of conventional SC. To lower the complexity, we consider a method of pruning the children binary tree which leaf nodes are all frozen bits or information bits under the chosen different parameters, according to the reliability of the polarized different channels. Simulation shows that the proposed scheme has a better decoding performance compared with successive cancellation decoding, and also reduces the time complexity of the successive cancellation decoding based on deep learning.*

Keywords: Polar codes, Successive cancellation decoding, Deep learning, Recurrent neural network, Long short-term memory network (LSTM)

1. **Introduction.** Polar codes [1] are a class of error correction codes that are mathematically proven to achieve channel capacity. Compared with the known LDPC and Turbo, polar codes have the properties of regular encoding structure, explicit construction method and low encoding and decoding complexities. Due to these advantages, polar codes have been chosen as coding scheme for the enhanced mobile broadband (eMBB) control channels of the 5th generation (5G) wireless communication systems since 2016. It is known that successive cancellation (SC) decoding and belief propagation (BP) decoding are both the basic decoding algorithms of polar codes. However, their decoding performance under finite-length polar codes still does not meet the requirement of 5G communication. To reduce the gap to capacity, some improved SC algorithms such as SC list (SCL) [2] are proposed, which can approach the maximum-likelihood (ML) performance with a larger list size. Moreover, when polar codes aided by CRC, the SCL decoder may successfully compete with some optimized LDPC codes. However, SCL decoding suffers from high latency and low throughput since its sequential decoding process is the same as that of SC. On the other hand, although BP can achieve higher throughput since its iterative message passing process can be executed in parallel, it still suffers from poor performance within limited iterations.

With the development of neural network, deep learning (DL) technologies have drawn much attention since they can provide promising performance in many tasks, such as speech recognition, game playing, machine translation and automatic driving [3,4]. In communication systems, DL has also been found in many applications at the physical

DOI: 10.24507/ijicic.17.03.789

layer functions, for example, channel coding, modulation recognition, physical layer security, channel estimation and equalization. In recent years, new decoding schemes based on deep-learning have shown to have good performance for decoding polar codes [5-16]. Among these different approaches, they can be mainly divided into two parts. The first one is to exploit DL based decoders to cope with the challenges of BP decoding algorithm [5-13]. The other part of research focuses on designing DL algorithms for SC decoding [14].

Specifically, the first one includes two baseline ideas. 1) Replace the conventional decoding algorithms completely with a fully-connected neural network. Although it can show near optimal performance with structured codes, the train complexity increases exponentially with block length. And also, this approach can cause the curse of dimensionality since the high dimensionality of codewords requires much more training data and degrades the ability of generation for unseen codewords [5]. 2) Leverage the well-developed BP algorithm via unfolding the iterative structure to the layered structure of neural network [6,10]. Simulation results show that the performance of the unfolded BP decoder with learned weights significantly outperforms traditional BP decoders [10]. Although this technique of DL is very powerful, it needs massive number of weights which results in considerable memory overhead for storage. On another side, it needs to perform massive multiplication for the weights updating on the neural networks, which leads to an increase in complexity and occupied area for hardware implementation [14]. In fact, we can learn from [10] that the unfolded BP decoder still requires high number of BP iterations to achieve a reasonable error correction performance as that of SC decoding which in turn decelerates the speed of decoding.

To overcome the above issues, an NN-based neural SC decoder that consists of multiple constituent NN decoders which are connected together using SC decoding is proposed as the second DL technology for polar codes. The approach has significantly smaller decoding latency while maintaining the same error correction performance when compared with [10] and original SC decoding.

In this paper, we propose an RNN-based neural SC decoder that consists of multiple constituent recurrent neural network (RNN) decoders which are more powerful NN models to further improve the decoding performance of [1]. Besides, to avoid the severely increased memory overhead and additional computations from DL model, we apply our domain knowledge in communication systems with carefully designed DL architecture. The main contributions of our work are described as follows.

1) We apply LSTM to the SC decoder, which can effectively extract the features from the sequential decoding process and enhance log-likelihood ratio (LLR) prediction of every node in the SC decoding binary tree. Therefore, the proposed design can improve the block error rate (BLER) performance and decoding latency by 0.11dB and 41.25%, respectively.

2) By taking advantage of domain knowledge for input data preprocessing and output dimension reduction, we can significantly reduce the memory requirements and computational complexity by over 20, making our design more feasible for hardware implementation.

The rest of this paper is organized as follows. Section 2 briefly reviews the concept of polar codes, DL, and the prior work. Section 3 illustrates the proposed architecture and main algorithms of LSTM-assisted SC with novel input data preprocessing and output design. The numerical experiments and analyses are shown in Section 4. Finally, Section 5 concludes our work.

2. Problem Statement and Preliminaries.

2.1. **Polar codes.** By recursively applying channel combining and splitting operations on N ($N = 2^n$, $n \geq 1$) independent copies of W, we can get a set of N polarized subchannels, denoted by $W_N^{(i)}\left(y_1^N, u_1^{i-1} \middle| u_i\right)$ where $i = 1, 2, \ldots, N$. The construction of an (N, K) polar code is completed by choosing the K best polarized subchannels which carry information bits and freezing the remaining subchannels to zero or some fixed values. According to the encoding rule of polar codes, the encoder can first generate the source block u_0^{N-1}. Then, polar coding is performed on the constraint $x_0^{N-1} = u_0^{N-1}\boldsymbol{G}_N$, where $\boldsymbol{G}_N$ is the generator matrix and the vector x_0^{N-1} is the codeword. The matrix $\boldsymbol{G}_N$ is obtained by the formula $\boldsymbol{B}_N\boldsymbol{F}^{\otimes n}$, where $\boldsymbol{F}^{\otimes n}$ is the n-th Kronecker power of $\boldsymbol{F} \triangleq \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ and $\boldsymbol{B}_N$ is the bit-reversal permutation matrix. The encoding process of an (N, K) polar code is shown as Figure 1.

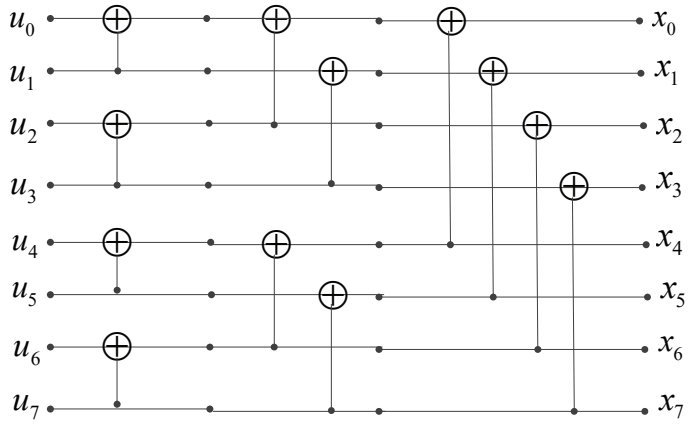

FIGURE 1. The encoding process of an (N, K) polar code

When the codeword x is modulated by using binary phase shift keying (BPSK) and transmitted by the additive white Gaussian noise (AWGN) channel, we can receive a set of initial channel values $\mathbf{y}$. It is known that the vector $\mathbf{y}$ can be computed by $\mathbf{y} = (1-2\mathbf{x})+\mathbf{z}$, where the noise vector $z \in \mathbf{R}^N$ and z is satisfied with $N(0, \sigma^2)$. By using the soft values $\mathbf{y}$, the receiver can compute the LLR of each bit in the codeword $\mathbf{x}$ to get the estimate vector $\hat{\mathbf{u}}$ of information source vector $\mathbf{u}$.

2.2. **Successive cancellation decoding of polar codes.** In [1], SC decoding is proposed as a baseline algorithm which has a very low complexity $N\log N$, where N is the codeblock length. Since the recursive structure of the polar encoder, the SC decoder performs a series of interlaced step-by-step decisions, which heavily depend on the decisions of the previous steps and the received sequence y_1^N from channel. The process of SC decoding can be divided into $(\log N + 1)$ stages as shown in Figure 2. Soft information $\alpha = \{\alpha_0, \alpha_1, \ldots, \alpha_{N-1}\}$ and bit estimated value $\beta = \beta_0, \beta_1, \ldots, \beta_{N-1}$ are transmitted recursively among these nodes between the $(\log N + 1)$ stages. Lastly, the estimations $\mathbf{u}$ of the information source corresponding to the N nodes in the leftmost stage of Figure 2 will be got.

In Figure 2, the root node of the binary tree firstly computes the soft-message values by using the channel initial LLRs, and then sends them to its left child. Following that, the decoding process of a binary-tree branch with root ν includes two phases [17].

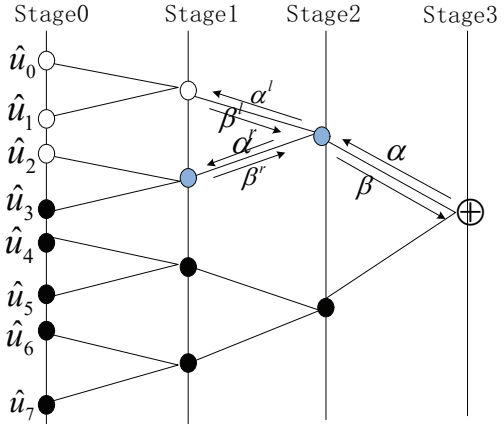

FIGURE 2. The SC decoding binary tree of an (N, K) polar code

1) Suppose node ν not to be a leaf node. If node ν is in stage d_ν, the soft message passed to its left child is computed as:

$$\alpha_{\nu_\ell}[i] = \alpha_\nu[2i] \triangle \alpha_\nu[2i+1] \tag{1}$$

where $i = 0 : 2^{\log N - d_\nu - 1} - 1$ and the binary operator $\triangle$ which combines two extended real numbers x and y is defined to form $x \triangle y := 2\arctan(\tanh(x/2)\tanh(y/2))$.

The left child ν_ℓ computes its hard decision value β_{ν_ℓ}, and passes β_{ν_ℓ} to its parent node ν. Then, node ν computes the soft information value α_{ν_γ} and sends it to its right child ν_γ.

$$\alpha_{\nu_\gamma}[i] = \alpha_\nu[2i](1 - 2\beta_{\nu_\ell}) + \alpha_\nu[2i+1] \tag{2}$$

where $i = 0 : 2^{\log N - d_\nu - 1} - 1$. The right child node ν_γ calculates the hard decision value β_{ν_τ} when it receives α_{ν_γ} and passes it to its parent node ν. The node ν calculates

$$\beta_\nu[2i+1] = \beta_{\nu_\gamma}[i], \quad \beta_\nu[2i] = \beta_{\nu_l}[i] \oplus \beta_{\nu_\gamma}[i] \tag{3}$$

2) If the node ν is a leaf node, it sets $\beta_\nu = 0$. Otherwise,

$$\beta_\nu = \begin{cases} 0 & \alpha_\nu \geq 0 \\ 1 & \alpha_\nu < 0 \end{cases} \tag{4}$$

2.3. **Deep neural network decoding.** DL has made remarkable achievements in the fields of computer vision and natural language processing and it has been adopted for application in channel decoding. The data-driven DL approach in [18] converted the decoding task into the pure idea of learning to decode by optimizing the general black box fully connected deep neural network (FC-DNN). The model can be seen as a simulator of a universal function $f : y = f(x_0; \theta)$ with network input $x_0 \in R$ and output $y \in R$, where θ is a set of parameters which can help to simulate the function f accurately. The structure of DNN includes input layer, multi-hidden layers and output layer as shown as Figure 3.

Each layer has many neurons z. When it is used in channel decoding, the vector lengths of input and output are both the codeblock. It is known that the original channel LLRs as input vector are fed to DDN and computed as $o_t = f_t(o_{t-1}) = \phi_t(o_{t-1}\omega_t + b_t)$, where $o_0 = \alpha$ and ϕ_t is a non-linear activation function such that

$$\phi_t(y) = \begin{cases} \max(0, y) & \text{if } 1 \leq t \leq T \\ \dfrac{1}{1+e^{-y}} & \text{if } t = T+1 \end{cases} \tag{5}$$

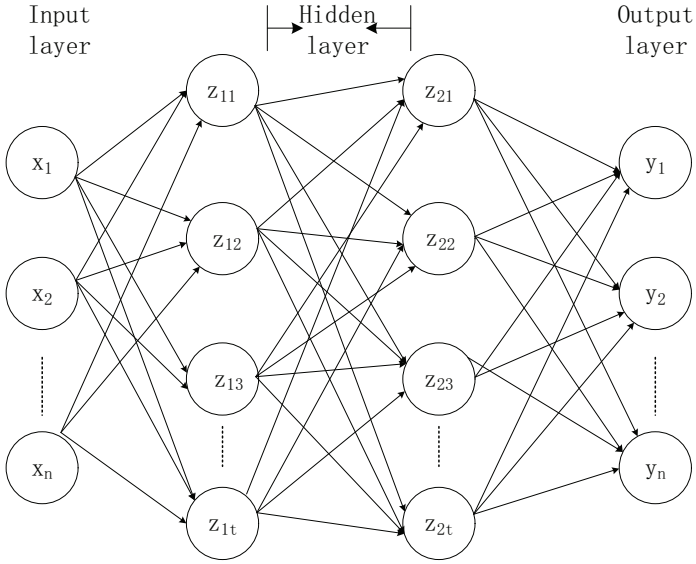

FIGURE 3. Network structure of DNN

The Sigmoid function $\phi(y) = \frac{1}{1+e^{-y}}$ can be as an activation function of output layer since it can make the values of output layer within the range of $(0, 1)$, which are considered as the probabilities of each bit being 0 or 1 of a codeword.

Weight and bias are two kinds of important parameters of DNN. Specifically, weight reflects the impact scale between neurons, while bias describes whether neurons are activated. To minimize the loss function, back propagation (BP) or stochastic gradient descent (SGD) is used to adjust the two kinds of parameters iteratively in training phase until DNN converges and becomes stable.

2.4. **Recurrent neural network decoding.** Although DNN-SC [10] and DNN-BP [14] show the property of low computation complexity or better performance, the deep architecture needs massive memory to store weights and also consumes a lot of energy when fetching the weights. Both of these issues will hinder the deployment of DNN-SC in real world communication systems. Therefore, we propose a recurrent neural network-based successive cancellation (RNN-SC) for polar decoder to share the weights among different iterations to reduce the memory overhead.

RNN is recurrent neural network and suitable for learning connected sequential data, in which the current output of a sequence is also related to the previous output. Specifically, RNN uses its internal memory to save the previous information and applies it to the calculation of the current output. Hence, it is very powerful for sequential signal processing. Considering the traditional RNN suffers from severe vanishing and exploding gradient problem, we use long-short term memory (LSTM) in Figure 4 as our practical RNN implementation.

Compared with the DNN-SC, the greatest difference of LSTM-SC is that the feed-forward architecture is transformed into a recurrent network, which forces the decoder to reuse weights among different iterations and results in a totally different optimization.

Though recurrent architecture effectively reduces the required number of parameters, the floating-point parameters still can hinder the hardware implementation of neural network decoder. Besides, the additional multiplications for scaling parameters also increase the computational complexity. Hence, codebook-based weight quantization [7] is proposed to effectively quantize the parameters and alleviate computational complexity without visible performance degradation.

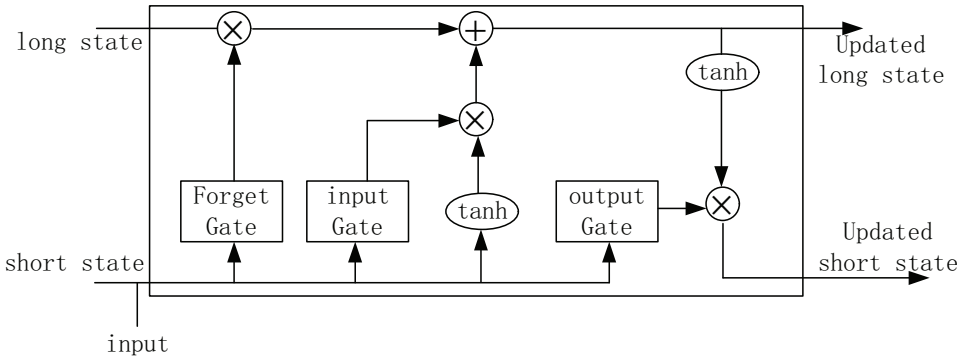

FIGURE 4. The architecture of LSTM

3. **Successive Cancellation Decoding Aided by LSTM.** Replace each constituent SC decoder of length 2^s at stage s in the SC decoding tree with an LSTM decoder. Then, four LSTM decoders are used to replace constituent SC decoders at stage 1 when the codeblock is 8 as Figure 5 shows.

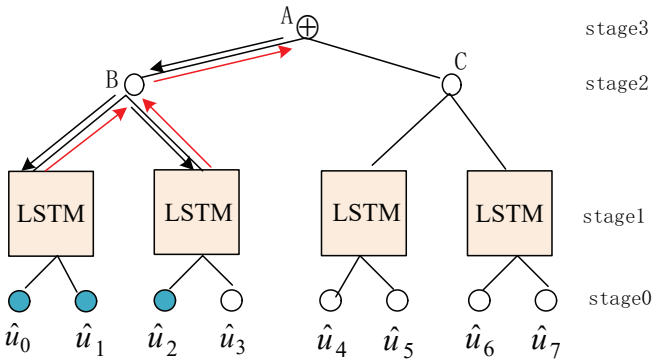

FIGURE 5. NSC decoding with RNN decoders applied at stage 1 of $P(8,5)$ with $(u_0, u_1, u_2) \in I$ polar code

In supervised learning, the scheme of recurrent successive cancellation decoding can be divided into two parts: one is recurrent network training and the other is SC decoding aided by LSTM. First, we should obtain the training data sequences so that the LSTM decoder can learn a mapping between input and output spaces. Each training sample pair (x_n, t_n) is generated from the same true joint distribution $p(x,t)$ and the sample pairs are independent identically distributed $(i.i.d)$. Then we can obtain a predictor which performs well on any possible relevant input x.

3.1. **Method of obtaining data sequences for training.** Assuming the SC decoder has already had the information of the transmitted bits, it can obtain the training data of each LSTM decoder. These training data includes hard values and corresponding internal LLR values. For the hard values of each node in the decoding tree, we can calculate them out from stage n to stage 0 only by using the message bits. While for the corresponding internal LLR values, they can be calculated out given that all the previous bits are decoded correctly. Based on the above analysis, we have the following two algorithms to obtain the two kinds of training data for the k-th LSTM decoder at stage s in the decoding tree.

We use Algorithm 1 to obtain hard training values for LSTM network based on the aforementioned Formula (3). In Algorithm 1, the variable d_ν denotes the current level number and is initialized to 1, which denotes to compute the hard values of each internal

node for training data starting from stage 1 to stage $\log N$. And c_2 is the previous level number of the current level in the decoding tree. The other variables such as c_1, c_3 and c_4 denote the nodes number in the current level or the previous level, respectively. In current level d_ν, it has $N/2^{d_\nu}$ nodes in total and needs to compute 2^{d_ν} hard values of each node m.

Algorithm 1 Algorithm of obtaining hard training values for LSTM

Input: the source bit sequences u;
Output: internal bits sequence β;
$d_\nu = 1$, $m = 0$, $nBase = \log N$;
use u to initialize part of two-dimensional vector β
while $d_\nu \leq nBase$ **do**
    $m = 0$;
    int c_1, c_2, c_3, c_4;
    while $m < N/pow(2, d_\nu)$ **do**
        for $i_1 < pow(2, d_\nu)$ **do**
            $c_1 = 2 * i_1 + m * pow(2, d_\nu)$;
            $c_2 = d_\nu - 1$;
            $c_3 = i_1 + m * pow(2, d_\nu) + pow(2, d_\nu - 1)$;
            $c_4 = i_1 + m * pow(2, d_\nu)$;
            $\beta[d_\nu][c_1] = (\beta[c_2][c_4] + \beta[c_2][c_3]) \oplus 2$;
            $\beta[d_\nu][c_1 + 1] = \beta[c_2][c_3]$;
        end for
        $m++$;
    end while
    $d_\nu++$;
end while

Algorithm 2 is used to obtain LLR training values for LSTM network based on the aforementioned Formulas (1) and (2). In Algorithm 2, the variable d_ν also denotes the current level number, but it is initialized to 1 which shows the computation of internal LLR training values of each node starting from stage $\log N - 1$ to stage 0. In each level, the decoder needs to compute 2^{d_ν} LLR values of each node m in $N/2^{d_\nu}$ nodes. And in Formula (1), the binary operator $\triangle$ is defined to combine two extended real numbers x and y to form $2\arctan(\tanh(x/2)\tanh(y/2))$. The function TanFunction() here is used to realize the operator $\triangle$ in Algorithm 3.

In Algorithm 3, the function TanFunction() first computes the tangent value of two input parameters, respectively. And then we can get the product of the two hyperbolic tangents. Before calculating the arctangent value of the product, it gives the solution method when the product is out-of-bounds. Lastly, the returned arctangent value can be obtained.

3.2. **LSTM-aided SC decoding.** The decoding process of SC aided by LSTM consists of two phases: one is from top level to the $stage_{index}$ level and the other is from the $stage_{index}$ level to the bottom leaf level. Specically, the $stage_{index}$ level is the defined level.

The LLR values obtained from the training phase are used as the input vector for the LSTM. The node number of the input layer depends on from which stage the decoding tree is classified into bottom part and top part. If we set the parameter $stage_{index}$ to j, the number of nodes in the input layer is 2^j. Hence, this parameter indicates from which

Algorithm 2 Algorithm of obtaining LLR training values for LSTM

Input: the original LLR vector L;
Output: the internal LLRs sequence α;
$d_\nu = nBase$;
use L to initialize part of two-dimensional vector α
while $(d_\nu \geq 0)$ **do**
  $m = 0$;
  $c_1 = 0$, $c_2 = 0$, $c_3 = 0$, $c_4 = 0$;
  while $m < N/pow(2, d_\nu)$ **do**
    for $i < pow(2, d_\nu - 1)$ **do**
      $c_1 = 2 * i + m * pow(2, d_\nu)$;
      $c_2 = d_\nu + 1$;
      $c_3 = i + m * pow(2, d_\nu) + pow(2, d_\nu - 1)$;
      $c_4 = i + m * pow(2, d_\nu)$;
      $\alpha[c_2][c_4] = \text{TanFunction}(\alpha[d_\nu][c_1], \alpha[d_\nu][c_1 + 1])$;
      $\alpha[c_2][c_3] = \alpha[d_\nu][c_1] * (1 - 2 * \beta[c_2][c_4]) + \alpha[d_\nu][c_1 + 1]$;
    end for
    $m++$;
  end while
  $d_\nu --$;
end while

Algorithm 3 The Function TanFunction(t_1, t_2)

Input: the updated LLRs t_1 and t_2 of the current node's parent node;
Output: an updated LLR t of current node;
$t = 0.0$, $t_5 = 0.0$, $t_3 = 0.0$, $t_4 = 0.0$;
$t_3 = \tanh(t_1/2)$;
$t_4 = \tanh(t_2/2)$;
$t_5 = t_3 * t_4$;
if $t_5 \geq 1$ **then**
  $t_5 = 0.9999999999999999$;
else
  if $t_5 \leq -1.0$ **then**
    $t_5 = -0.9999999999999999$;
  end if
end if
$t = 2 * a\tanh(t_5)$
return t;

levels of the decoding tree will be replaced by LSTM decoders to complete the remaining decoding process. The following Algorithm 4 shows the SC decoding process aided by LSTM.

In Algorithm 4, the indicator array $\mathbf{a}[d_\nu]$ is equal to 0, 1 or 2, which corresponds to the current node operation of the d_ν-th level in the decoding tree. If $\mathbf{a}[d_\nu] = 0$, the current node should compute LLR values and pass them to its left children. If $\mathbf{a}[d_\nu] = 1$, the current node computes and passes the LLR values to its right children. If $\mathbf{a}[d_\nu] = 2$, the current node should compute its internal hard values and pass them to its parents. Since the decoding process in the decoding tree is performed according to depth-first rule, the

Algorithm 4 The process of SC decoding aided by LSTM

import all kinds of required packages;
read CSV file converted from txt file to the data set;
Input: the received channel sequence y_0^{N-1}, two indicator arrays **a** and **m**, the index set of chosen bit **chosenI**;
Output: bit estimated values **uu**;
$d_\nu = \log N$;
initialize indicator arrays **a** and **m** to be 0

while $(a[\log N] \leq 2)$ **do**
  while $d_\nu > stage_{index}$ **do** //perform SC decoding
    if $a[d_\nu] == 0$ **then** //compute $2^{d_\nu - 1}$ LLRs passed to the left children
      $c_1 = d_\nu - 1$;
      for $i_3 < pow(2, d_\nu - 1)$ **do**
        $c_2 = i_3 + m[d_\nu] * pow(2, d_\nu)$;
        $c_3 = 2 * i_3 + m[d_\nu] * pow(2, d_\nu)$;
        $LLR[c_1][c_2] = TanFunction(LLR[d_\nu][c_3], LLR[d_\nu][c_3 + 1])$;
      end for
      $a[d_\nu] + +$;
      $d_\nu - -$;
    end if
    if $a[d_\nu] == 1$ **then** //compute $2^{d_\nu - 1}$ LLRs passed to the right children
      $c_1 = d_\nu - 1$;
      for $i_4 < pow(2, d_\nu - 1)$ **do**
        $c_2 = i_4 + m[d_\nu] * pow(2, d_\nu)$;
        $c_3 = 2 * i_4 + m[d_\nu] * pow(2, d_\nu)$;
        $c_4 = i_4 + m[d_\nu] * pow(2, d_\nu) + pow(2, d_\nu - 1)$;
        $LLR[c_1][c_4] = LLR[d_\nu][c_3] * (1 - 2 * bit[c_1][c_2]) + LLR[d_\nu][c_3 + 1]$;
      end for
      $a[d_\nu] + +$;
      $d_\nu - -$;
    end if
    if $a[d_\nu] == 2$ **then** //compute 2^{d_ν} internal bit values of the current node
      $c_1 = d_\nu - 1$;
      for $i_5 < pow(2, d_\nu - 1)$ **do**
        $c_2 = 2 * i_5 + m[d_\nu] * pow(2, d_\nu)$;
        $c_3 = i_5 + m[d_\nu] * pow(2, d_\nu)$;
        $c_4 = i_5 + m[d_\nu] * pow(2, d_\nu) + pow(2, d_\nu - 1)$;
        $bit[d_\nu][c_2] = (bit[c_1][c_3] + bit[c_1][c_4])\%2$;
        $bit[d_\nu][c_2 + 1] = bit[c_1][c_4]$;
      end for
      if $d_\nu == nBase$ **then**
        $a[d_\nu] + +$;
      else
        $a[d_\nu] = 0$;
      end if
      $m[d_\nu] + +$;
      $d_\nu + +$;
    end if
  end while

```
    if d_ν == stage_index then        //reaching the defined level
        LSTMDecoder();        //perform LSTM decoder for the remaining stages
        c_1 = d_ν − 1;        //compute 2^{d_ν−1} bit values of the current node
        for i_6 < pow(2, d_ν − 1) do
            c_2 = 2 ∗ i_6 + m[d_ν] ∗ pow(2, d_ν);
            c_3 = i_6 + m[d_ν] ∗ pow(2, d_ν);
            c_4 = i_6 + m[d_ν] ∗ pow(2, d_ν) + pow(2, d_ν − 1);
            bit[d_ν][c_2] = (bit[c_1][c_3] + bit[c_1][c_4])%2;
            bit[d_ν][c_2 + 1] = bit[c_1][c_4];
        end for
        m[d_ν] + +;
        d_ν + +;
    end if
end while
```

value $\mathbf{a}[d_\nu]$ can vary with different nodes of the d_ν-th level. Another indicator array $\mathbf{m}$ is used to show which value of a node should be computed. And the two-dimensional arrays **LLR** and **bit** are used to store LLR values and internal hard values of each node, respectively.

When the SC decoder reaches the defined $stage_{index}$-th level, the SC decoding process stops. And the program calls the function $LSTMDecoder()$ and $2^{\log N - stage_{index}}$ LSTM decoders begin to work. An LSTM decoder corresponding to a node in the $stage_{index}$-th level uses $2^{stage_{index}}$ LLRs of the node as its input and computes $2^{stage_{index}}$ internal hard values of the node as its output.

This network is mainly composed of one LSTM layer, two fully connected layers, a dropout layer and an output layer. There are $2^{stage_{index}}$ timesteps in each LSTM layer. And each timestep contains t_n hidden LSTM unit. The variable t_n is set to 220 and the ratio of dropout is 0.05 in our simulation. The function $relu()$ is as the activate function of hidden layer to strengthen the non-linear relationship between these layers, and the output layer uses the activate function $Sigmoid()$ to normalize the values of LSTM to a range of (0.0-1.0). In view of the property of the concave and protruding of the function, logarithmic function is used to avoid local shock during gradient descent optimization. The specific algorithm is as Algorithm 5.

Each LSTM network takes $2^{stage_{index}}$ LLR sequences of each corresponding node as input and outputs a $2^{stage_{index}}$-dimension vector which is determined by the probability calculated by the $Sigmoid()$ function in the output layer. With the LSTM architecture, the network produces multiple outputs and we use the mean-squared-error (MSE)

$$\zeta_{MSE} = \frac{1}{M} \sum_{j=0}^{M-1} \sum_{i=0}^{N_v-1} \left(\mu_i^{(j)} - P_i^{(j)} \right)^2 \tag{6}$$

as the loss function, where $\mu_i^{(j)}$ and $P_i^{(j)}$ are the bit value and the output probability of the i-th bit in the j-th message word of the min batch, respectively. Then stochastic gradient descent is used to minimize the loss.

To train the networks, we need to generate a training set, and each sample in the training set contains the values of LLR sequences and corresponding internal hard values. We carry out extensive simulations to acquire the training set. First, random source information sequences are generated and codewords are obtained by polar coding. Then, after BPSK modulation, we can receive the data sequences polluted by AWGN channel noise. SC decoding is used to correct the errors of the received data. In the binary SC

Algorithm 5 LSTMDecoder()

import all kinds of required packages;
read CSV file converted from txt file to the data set;
Input: parameters of input and hidden layers *weights*, *biases*;
Output: values to (0.0-1.0) of output layer;
function $multilayer\_perception(X1, weights1, biases1)$
    $relu()$; //compute the values of hidden layer
    $sigmoid()$; //compute the values of output layer
end function
pred = $multilayer\_perception(\_x, \_weights, \_biases)$;
$cross\_entropy = -(y \times tf.\log(pred) + (1-y) \times tf.\log(1-pred))$;
$loss = reduce\_mean(cross\_entropy)$;
$train\_1 = tf.train.GradientDescentOptimizer(learning\_rate = 0.001).$
$minimize(loss)$;
$training\_epochs = 100$;
$minibatchSize = 100$;
with tf.Session() as sess:
initialized all variables;
$saver = tf.train.Saver()$;
for epoch in range ($training\_epochs$) **do**
    for i in range (np.int32($len(L\_result)/minibatchSize$)) **do**
        x1 = $L\_result[i \times minibatchSize : (i+1) \times minibatchSize]$;
        y1 = $b\_result[i \times minibatchSize : (i+1) \times minibatchSize]$;
        _, lossval, errval=sess.run([train_1, loss, pred], feed_dict=x:x1, y:y1);
        $summerr = summerr + lossval$;
    end for
end for

tree, we store LLR sequence and hard values of each internal node of the defined stage as a sample in the training set.

3.3. **Optimization of LSTM-aided SC decoding.** From above, we can see that the whole decoding process based on a binary tree can be divided into two parts: top region and bottom region. The nodes in the top part perform the original SC decoding, while others in the bottom are divided into several parts and perform different LSTM decoding respectively. Figure 5 shows that the SC decoding binary tree of a $(8, 5)$ polar code has four stages. If $stage_{index} = 1$, stage 3 and stage 2 are both in top part and the remaining stages are in bottom part.

The sequential structure of SC decoding makes the decoding process proceed by traversing the binary tree to visit the nodes at level 0 from left to right. While for NSC algorithm, it does not fully consider the parallel decoding of special nodes in the SC decoding tree and limits the reduction of system delay to a certain extent. To further reduce latency and improve low throughput, several types of nodes are considered in the decoding process. Specifically, Rate-0 nodes and the hard bit estimations can be directly calculated at level s where the Rate-0 node is located, because the values of frozen bits are known to the decoder. And also, there is no need to traverse the decoding tree below Rate-1 nodes, since the hard bit estimations can be directly calculated at level s where the Rate-1 node is located. Hence, the branch of binary tree corresponding to Rate-1 nodes or Rate-0 nodes can be directly pruned. On another side, the last child node of the node Rep is

information bit, and the other child nodes are frozen bits [19]. If the node Rep has q-child nodes, the previous $q-1$ hard decision β_ν of Rep are all 0. The q-th hard decision depends on $\sum_{i=1}^{q} \alpha_\nu[i]$. If the value $\sum_{i=1}^{q} \alpha_\nu[i] \geq 0$, then $\beta_\nu^q = 0$, or else $\beta_\nu^q = 1$. We can use these three kinds of nodes to simplify the SC decoding process and change the serial decoding to partial parallel decoding. And the number of LSTM decoders also can be reduced. According to above analysis, in Figure 6, we only need three LSTM decoders to replace the original branches of SC decoding tree at stage 3. The branches corresponding to Rate-0 codes and Rate-1 codes can be directly pruned. The optimization algorithm of LSTM-aided SC decoding is as Algorithm 6.

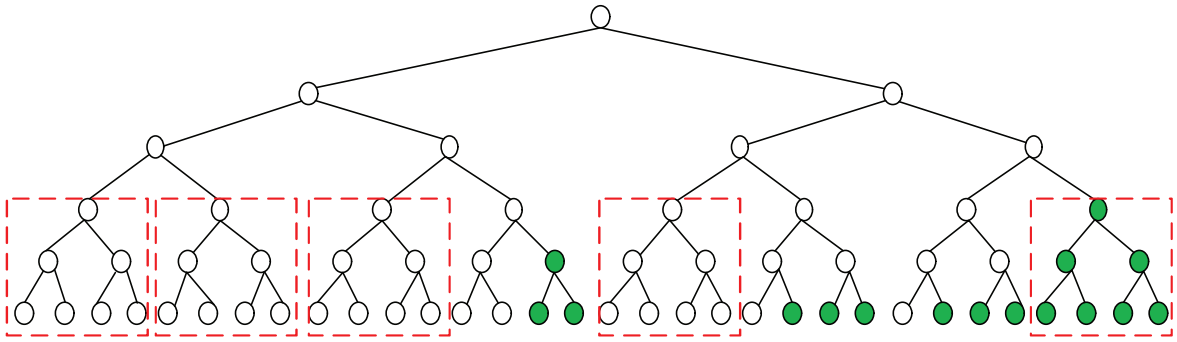

FIGURE 6. SC decoding tree of $(32, 12)$ polar code with $\left(u_{14}, u_{15}, u_{21}^{23}, u_{25}^{31}\right) \in I$

Algorithm 6 SC decoding LSTM-aided of polar codes

Input: the received channel sequence y_0^{N-1}, two signed arrays a and m, the chosen bit_{index} values $chosenI$;
Output: bit estimated values uu;
$d_\nu = \log N$;
$leaf_{index} = 0$;
while $(a[\log N] \leq 2)$ **do**
  while $d_\nu > stage_{index}$ **do** //perform SC decoding when not reaching the defined level
    if $a[d_\nu] == 0$ **then** //compute LLR values passed to the left children of current node
      compute $2^{d_\nu - 1}$ LLRs;
      $a[d_\nu] + +$;
      $d_\nu - -$;
    else
      if $a[d_\nu] == 1$ **then**
        compute $2^{d_\nu - 1}$ LLRs passed to the right children;
        $a[d_\nu] + +$;
        $d_\nu - -$;
      end if
      if $a[d_\nu] == 2$ **then**
        compute 2^{d_ν} bit values of the current node
        if $d_\nu == nBase$ **then**
          $a[d_\nu] + +$;
        else
          $a[d_\nu] = 0$;

end if
$m[d_\nu] + +$;
$d_\nu - -$;
end if
end if
end while
if $d_\nu == stage_{index}$ **then** //reaching the defined level
if $d_\nu \in R\_0$ **then** //directly compute LLR values to its right child
compute $2^{d_\nu - 1}$ LLRs passed to the right children;
$a[d_\nu] + +$;
$d_\nu - -$;
else
if $d_\nu \in R\_1$ **then** //directly compute LLR values to its right child
compute $2^{d_\nu - 1}$ LLRs passed to the right children;
$a[d_\nu] + +$;
$d_\nu - -$;
else
$LSTMDecoder()$; //perform LSTM decoder
compute $2^{d_\nu - 1}$ bit values of the current node
$m[d_\nu] + +$;
$d_\nu + +$;
end if
end if
end if
end while

4. **Numerical Example.** In this section, we consider a $(128, 64)$ rate-1/2 polar code C_1 to examine the error correction performance of the proposed LSTM decoder in terms of frame error rate (FER). All simulations are performed over an additive white Gaussian noise (AWGN) channel with binary phase shift keying (BPSK).

4.1. **Choice of $stage_{index}$.** In the above algorithms, the variable $stage_{index}$ denotes the number of levels at which LSTM decoders are used to replace SC decoders to perform the remaining decoding process. Most references set the value according to the experience. Following the idea of [14], we list the values from 0 to $\log N$ and choose a value under which the decoder has almost the same or better performance compared with the original SC decoder.

4.2. **Comparison of the FER performance.** Simulation results show that the decoding performance and latency of polar codes partly depend on the value of variable $stage_{index}$ when using neural network-aided SC. Generally, the larger the value of $stage_{index}$, the better the performance and latency. However, $stage_{index}$ has a threshold value. Once larger than the threshold value, $stage_{index}$ has almost no influence on the performance and latency. For 128-length and 256-length polar codes, threshold values of $stage_{index}$ are both 4. So the scheme needs $2^7/2^4 = 8$ LSTM decoders for a $(128, 64)$ polar code and $2^8/2^4 = 16$ LSTM decoders for a $(256, 128)$ polar code, respectively. The performances of the two codes under different decoding schemes are shown as Figure 7 and Figure 8. We can see that the proposed scheme has better performance.

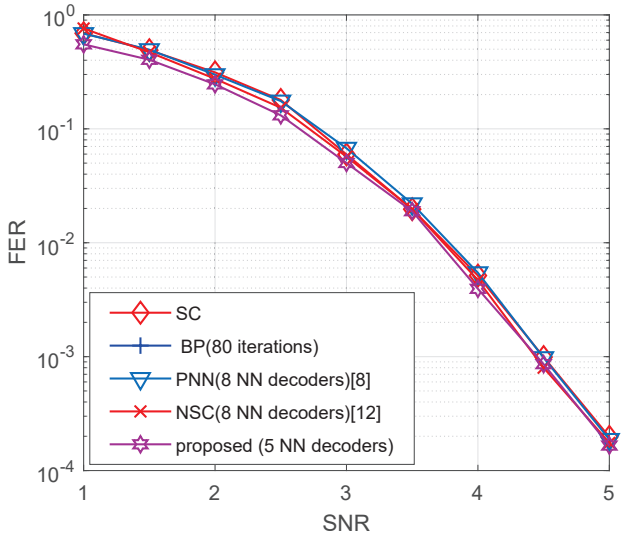

FIGURE 7. FER of a $(128, 64)$ polar code under SC, BP, PNN, NSC and the proposed method

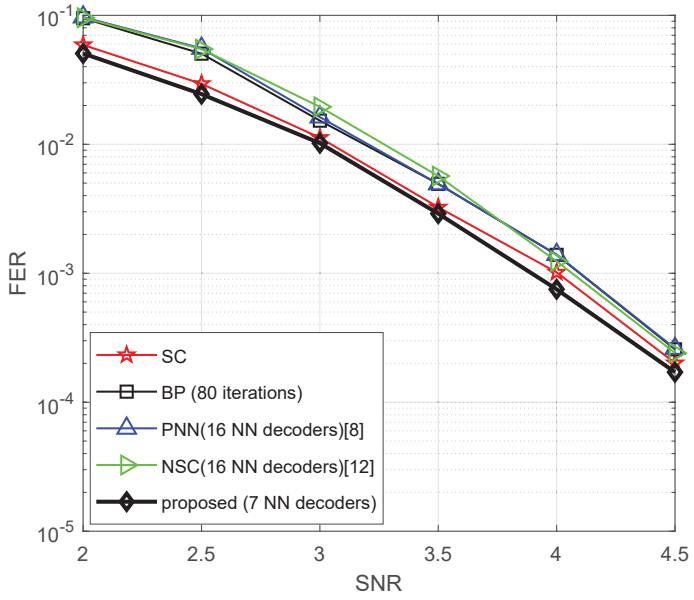

FIGURE 8. FER of a $(256, 128)$ polar code under SC, BP, PNN, NSC and the proposed method

4.3. **Comparison of average time complexity.** For an N-length polar code, its time complexities under different decoding schemes are different. Considering that lots of variables are used to denote these complexity formulas, we first provide a variable list as shown in Table 1 to make the used variables more clearly.

Specifically, BP algorithm scales much better with $O(\log N)$ and depends on the number of iterations T. Since it can be synchronization after each BP stage, namely, its time complexity is $2T \log N$. The PNN decoder enforces less BP updates and the NNDs itself only synchronize after each layer. Hence, it has $\frac{N}{N_p} \times N_H + \frac{N}{N_p} \times 2 \log \frac{N}{N_p}$ time complexity. The decoding latency in terms of the number of time steps for the NSC decoder can be calculated as $\frac{N}{2^{stage_{index}}} \times (N_H + 1) + 2 \times \frac{N}{2^{stage_{index}}} - 2$. Time complexities of these schemes are shown as Table 2.

TABLE 1. Symbol definition

Symbol	Meaning
N	the length of code block
T	iteration number of BP decoding
N_p	the size of partition
N_H	the number of hidden layer
q	the number of three special nodes at stage $stage_{index}$
q_0	the number of rate-0 at stage $stage_{index}$

TABLE 2. Time complexities under different decoding schemes for an N-length polar code

SNR	Time complexity
BP decoding	$2T \log N$
SC decoding	$2N$
PNN	$\frac{N}{N_p} \times N_H + \frac{N}{N_p} \times 2 \log \frac{N}{N_p}$
NSC	$\frac{N}{2^{stage_{index}}} \times (N_H + 1) + 2 \times \frac{N}{2^{stage_{index}}} - 2$
Proposed	$\left(\frac{N}{2^{stage_{index}}} - q\right) \times (N_H + 1) + 2 \times \frac{N}{2^{stage_{index}}} - 2 + q - q_0$

For NSC decoders in the above table, its time complexity in a T-hidden layer of network is $\frac{N}{2^{stage_{index}}} \times (N_H + 1)$. The time complexity of top part nodes in the SC decoding tree is $2 \times \frac{N}{2^{stage_{index}}} - 2$. So, the total time complexity of NSC is $\frac{N}{2^{stage_{index}}} \times (N_H + 1) + 2 \times \frac{N}{2^{stage_{index}}} - 2$. The proposed scheme only needs $\frac{N}{2^{stage_{index}}} - q$ LSTM decoders, where q is the number of special nodes at the stage $stage_{index}$. Its final time complexity is $\left(\frac{N}{2^{stage_{index}}} - q\right) \times (N_H + 1) + 2 \times \frac{N}{2^{stage_{index}}} - 2 + (q - q_0) \times 1$. As we know, Rate-0 node does not consume time step, while both Rep node and Rate-1 node only need one time step in the calculation process. So, $(q - q_0)$ denotes the number of time steps contained in these special nodes at the stage $stage_{index}$. For example, the index set of frozen bits for a $(128, 64)$ polar code is 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 48, 49, 50, 52, 56, 64, 65, 66, 67, 68, 69, 70, 72, 73, 74, 76, 80, 81, 82, 84, 96. When $stage_{index}$ is set to be 4, NSC needs 8 NN decoders but 5 for the proposed scheme. While $stage_{index} = 3$, the former and latter need 16 and 7 NN decoders, respectively. Comparisons of the time complexity between different decoding schemes are shown in Figure 9 and Figure 10.

From the two figures, it shows that the proposed scheme has much lower time complexity than other schemes at low stages, which give us a new thought of choosing appropriate stage to combine SC decoding with LSTM decoders.

5. **Conclusions.** In this paper, we proposed an LSTM-based successive cancelation decoding scheme to improve the FER performance and lower the decoding complexity. A pruning method combined with the LSTM decoding scheme and how to set the value of some important parameters are considered. In the end, simulation results are presented. However, the proposed scheme is only friendly to the short-length polar codes. In the future, we will research into the design for supporting moderate-length polar codes and investigate the decoding capability of the more powerful NN models.

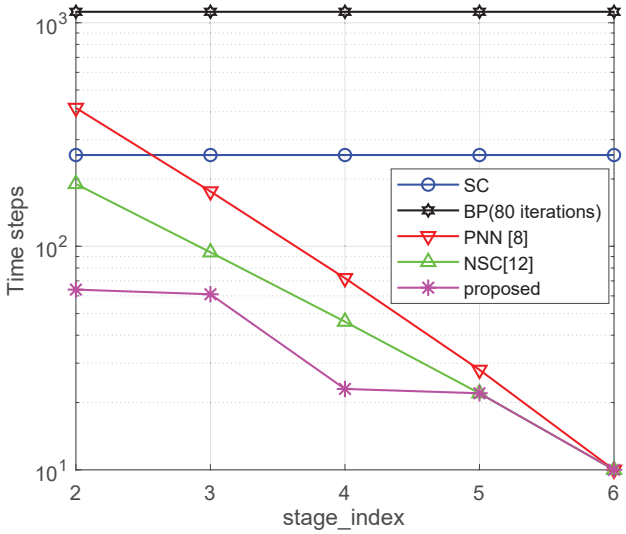

FIGURE 9. Comparison of time steps consumed of a $(128, 64)$ polar code under SC, BP, PNN, NSC and the proposed method

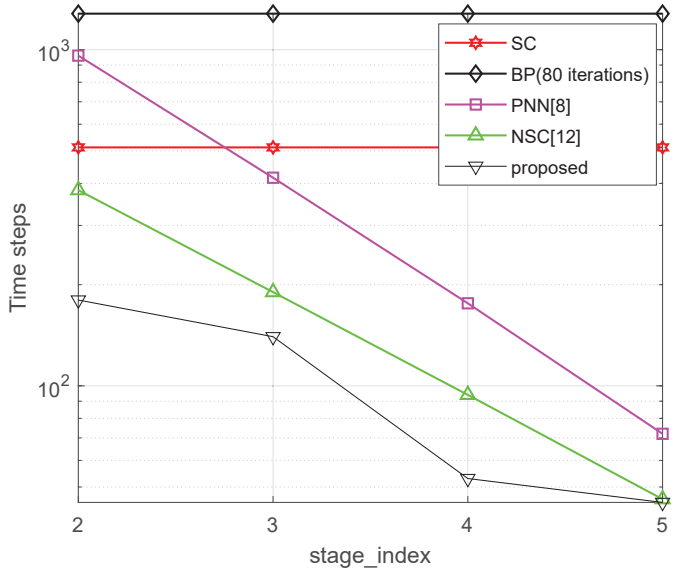

FIGURE 10. Comparison of time steps consumed of a $(256, 128)$ polar code under SC, BP, PNN, NSC and the proposed method

Acknowledgment. The authors would like to thank the editor and anonymous reviewers for their constructive comments which helped improve the quality of this paper. This work was supported in part by the Project Supported by Natural Science Basic Research Plan in Shaanxi Province of China (No. 2018JM6102), in part by the Principal Fund of Xi'an Technological University (No. XAGDXJJ17018), in part by the State and Provincial Joint Engineering Lab of Advanced Network, Monitoring and Control of Xi'an Technological University.

REFERENCES

[1] E. Arikan, Channel polarization: A method for constructing capacity achieving codes for symmetric binary-input memoryless channels, *IEEE Transactions on Information Theory*, vol.55, no.7, pp.3051-3073, 2009.

[2] I. Tal and A. Vardy, List decoding of polar codes, *IEEE Transactions on Information Theory*, vol.61, no.5, pp.2213-2226, 2015.
[3] K. Fithriasari and U. S. Nuraini, Face identification using multi-layer perceptron and convolutional neural network, *ICIC Express Letters*, vol.15, no.2, pp.157-164, 2021.
[4] C. Lee and B.-D. Lee, Enhancement for automatic extraction of RoIs for bone age assessment based on deep neural networks, *ICIC Express Letters*, vol.14, no.2, pp.163-170, 2020.
[5] T. Gruber, S. Cammerer, J. Hoydis and S. T. Brink, On deep learning-based channel decoding, *Proc. of IEEE the 51st Conference on Information Sciences and Systems*, Baltimore, pp.1-6, 2017.
[6] W. Lyu, Z. Zhang, C. Jiao, K. Qin and H. Zhang, Performance evaluation of channel decoding with deep neural networks, *Proc. of IEEE International Conference on Communications*, Kansas City, pp.1-6, 2018.
[7] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein and Y. Be'ery, Deep learning methods for improved decoding of linear codes, *IEEE Journal of Selected Topics in Signal Processing*, vol.12, no.1, pp.119-131, 2018.
[8] W. Xu, Z. Wu, Y.-L. Ueng, X. You and C. Zhang, Improved polar decoder based on deep learning, *Proc. of IEEE International Workshop on Signal Processing Systems*, Cape Town, pp.1-6, 2017.
[9] C.-F. Teng, C.-H. D. Wu, A. K.-S. Ho and A.-Y. A. Wu, Low complexity recurrent neural network-based polar decoder with weight quantization mechanism, *Proc. of the IEEE International Acoust., Speech, Signal Process.*, Brighton, pp.1413-1417, 2019.
[10] S. Cammerer, T. Gruber, J. Hoydis and S. ten Brink, Scaling deep learning-based decoding of polar codes via partitioning, *Proc. of IEEE Global Communication Conference*, Paris, pp.1-6, 2017.
[11] E. Nachmani, Y. Be'ery and D. Burshtein, Learning to decode linear codes using deep learning, *Proc. of the Annual Allerton Conference on Communication, Control and Computing*, Monticello, pp.341-346, 2016.
[12] L. Lugosch and W. J. Gross, Neural offset min-sum decoding, *Proc. of the IEEE International Symposium on Information Theory*, Aachen, pp.1361-1365, 2017.
[13] I. Be'ery, N. Raviv, T. Raviv and Y. Be'ery, Active deep decoding of linear codes, *IEEE Transactions on Communications*, vol.68, no.2, pp.728-736, 2020.
[14] N. Doan, S. A. Hashemi, M. Mondelli and W. J. Gross, Neural successive cancellation decoding of polar codes, *Proc. of IEEE International Workshop on Signal Processing Advances in Wireless Communications*, Kalamata, pp.1-5, 2018.
[15] J. Fang, M. Bi, S. Xiao, H. Yang, Z. Chen, Z. Liu and W. Hu, Neural successive cancellation polar decoder with tanh-based modified LLR over FSO turbulence channel, *IEEE Photonics Journal*, vol.12, no.6, pp.1679-1689, 2020.
[16] N. Doan, S. A. Hashemi, F. Ercan et al., Neural successive cancellation flip decoding of polar codes, *Journal of Signal Processing Systems*, vol.12, no.6, pp.1-12, 2020.
[17] C. Leroux, A. J. Raymond, G. Sarkis and W. J. Gross, A semi-parallel successive-cancellation decoder for polar codes, *IEEE Transactions on Signal Processing*, vol.61, no.2, pp.289-299, 2013.
[18] O. Simeone, A very brief introduction to machine learning with applications to communication systems, *IEEE Transactions on Cognitive Communications and Networking*, vol.4, no.4, pp.648-664, 2018.
[19] G. Sarkis, P. Giard, A. Vardy, C. Thibeault and W. J. Gross, Fast polar decoders: Algorithm and implementation, *IEEE Journal on Selected Areas in Communications*, vol.32, no.5, pp.946-957, 2014.