applied sciences

MDPI

Article

Application of Tensor Decomposition to Reduce the Complexity of Neural Min-Sum Channel Decoding Algorithm

Qingle Wu [1], Benjamin K. Ng [1,*], Yuanhui Liang [1], Chan-Tong Lam [1] and Yan Ma [1,2]

[1] Faculty of Applied Sciences, Macao Polytechnic University, Macao 999078, China
[2] BUPT Network Information Center, Beijing University of Posts and Telecommunications, Beijing 100876, China
* Correspondence: bng@mpu.edu.mo

Abstract: Channel neural decoding is very promising as it outperforms the traditional channel decoding algorithms. Unfortunately, it still faces the disadvantage of high computational complexity and storage complexity compared with the traditional decoding algorithms. In this paper, we propose that low rank decomposition techniques based on tensor train decomposition and tensor ring decomposition can be utilized in neural offset min-sum (NOMS) and neural scale min-sim (NSMS) decoding algorithms. The experiment results show that the proposed two algorithms achieve near state-of-the-art performance with low complexity.

Keywords: tensor train decomposition; tensor ring decomposition; neural offset min-sum; neural scale min-sum

check for updates

Citation: Wu, Q.; Ng, B.K.; Liang, Y.; Lam, C.-T.; Ma, Y. Application of Tensor Decomposition to Reduce the Complexity of Neural Min-Sum Channel Decoding Algorithm. *Appl. Sci.* **2023**, *13*, 2255. https://doi.org/10.3390/app13042255

Academic Editor: Andrea Prati

Received: 31 December 2022
Revised: 2 February 2023
Accepted: 2 February 2023
Published: 9 February 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Driven by breakthroughs of deep learning (DL) in image processing, natural language processing and other fields, researchers have proposed channel decoding algorithms based on DL in recent years. DL-based decoding has now become an active area of research. In [1], neural belief propagation (BP) for linear codes was proposed and then the authors applied recurrent neural network (RNN) to linear block codes [2]. Improving decoding performance by adding appropriate weight parameters and loss functions to the decoding algorithm, where the loss function is used to train these weight parameters, is called the model-driven decoding algorithm. The offset min-sum (OMS) decoding algorithm is an excellent decoding method with low complexity and high performance [3], and is widely used in Hamming codes, BCH and low-density parity check (LDPC) codes. However, there is still room for performance improvement in these schemes. Lugosch and W. J. G proposed a neural offset min-sum (NOMS) decoding algorithm in 2017. The authors in [4] proposed the model-driven deep learning neural OMS decoding algorithm for LDPC code, and proposed the neural share offset min-sum (NSOMS) decoding algorithm with lower complexity. In [5–7], both multiplicative weights and offset weights are used in neural scale min-sim (NSMS), which needs more storage space and computational complexity.

The model-driven BP decoding algorithm achieves state-of-the-art performance and outperforms traditional BP decoder algorithm by adding weights at appropriate locations, which also causes an increase in storage space and computational complexity. However, model-driven BP makes practical implementation challenging. These issues can be solved by model compression techniques. Low rank decomposition is one such model compression technique. Fully connected deep neural network (DNN) parameters are converted into tensor train format through tensor train (TT) decomposition [8]. In [9], it achieves seven times the compression while obtaining a state-of-the-art performance, which converts the fully connected layer to the tensorizing layer. In [10–13], they converted the dense weight matrices of the fully connected layers and convolutional layers to the tensorizing layer. Recurrent layers in [14,15] are converted to TT-Recurrent layers. Since tensor decomposition

is highly dependent on rank, it is difficult to find the optimal rank that minimizes the error of the product of the decomposed tensors and the original tensor. However, the restricted setting of ranks and restricted ordering of core tensors make TT-based compression models sensitive to parameter selection.

The authors of [16] proposed tensor ring (TR) decomposition, which satisfies the circular dimensional permutation invariance. The authors of [17] proposed a TR-based long short-term memory (TR-LSTM) model to reconstruct the input-to-hidden transformation by exploiting tensor ring decomposition, which achieved higher accuracy and higher compression rate than the TT-based long short-term memory (TT-LSTM). TR networks are proposed in [18]. The authors of [19] proposed nonlinear tensor ring networks (NTRN) to compress fully linked neural networks and convolutional neural networks (CNN), which adds a nonlinear activation function after the convolution operation inside the compression layer. TT-based neural normalized min-sum (TT-NNMS) was proposed in [20], which reduces the number of parameters and multiplications. However, ref. [20] only performed TT decomposition for the NNMS decoding algorithm, and did not perform TT decomposition for NOMS. The NOMS algorithm is different from the NNMS algorithm, and the offset addition replaces the factor multiplication in the neural normalized min-sum (NNMS) algorithm. In addition, ref. [20] only applied TT decomposition to the decoding algorithm of LDPC codes, and does not apply it to BCH and Hamming codes.

In this work, methods to reduce the complexity of NOMS and the NSMS decoding algorithm are proposed, which involve using TT decomposition or TR decomposition, and apply it to the decoding of Hamming code, BCH codes and LDPC code. The novelty of this paper is summarized as follows:

- As a continuation of the work in paper [20], we apply the TT decomposition to the NOMS and NSMS decoding algorithms, called the TT-NOMS and TT-NSMS algorithms, which can reduce the complexity of the algorithm.
- We apply the TR decomposition to the NOMS and NSMS decoding algorithms, and achieve the same compression effect as TT, but TR decomposition has more flexible choices in rank.
- In addition, we apply the TT-NOMS, TT-NSMS, TR-NSMS, and TR-NOMS algorithms to the decoding of Hamming code, BCH codes and LDPC code, which reduces the complexity of these decoding algorithms, while the performance is almost not degraded.

The remainder of this paper is organized as follows: OMS and SMS, NOMS and NSMS, TT decomposition and TR decomposition are described briefly in Section 2. In Section 3, we propose the TT-NOMS and TT-NSMS decoding algorithms. TR-NOMS and TR-NSMS decoding algorithms are proposed in Section 4. Simulations results and the complexity comparison are shown in Section 5. Section 6 shows the performance evaluation with NOMS, NSMS, TT-NOMS, TT-MSMS, TR-NOMS and TR-NSMS algorithm. Section 7 concludes the paper.

2. Preliminary

2.1. Offset Min Sum Decoder and Scale Min Sum Decoder

We consider (N,K) LDPC, BCH and Hamming code where N denotes the length of codeword, K denotes the number of information bits in the message. We use binary phase shift keying (BPSK) modulation, where the check matrix is H. The message X is encoded by generator matrix G. The message obtained is $Y = X + Z$ over the additive white gaussian noise (AWGN) channel, where $Z \in \mathbb{R}^n$ is the channel noise. The log-likelihood ratio (LLR) of the v-th bit can be computed as

$$l_v = log(\frac{P(Y_v = y_v|X_v = -1)}{P(Y_v = y_v|X_v = +1)}) = \frac{2y_v}{\sigma^2} \quad (1)$$

where $\sigma^2 = N_0/2$ is the variance of the channel noise, N_0 is noise spectral density, and we use $E_b/N_0 = 1/(2(K/N)\sigma^2)$ as the signal-to-noise ratio, where E_b is the energy per information bit, y_v is the v-th value of the received message.

The decoding graph is a bipartite graph, which consists of variable nodes (VNs) and check nodes (CNs). LLRs l_v are transmitted from the VNs $u^t_{v,c}$ to the CNs $u^t_{c,v}$, and then return back. Each iteration requires the following steps:

Step 1: From the VNs to CNs the message $u^t_{v,c}$ is passed in the graph as.

$$u^t_{v,c} = l_v + \sum_{c' \in N(v)\backslash c} u^{t-1}_{c',v}. \tag{2}$$

where the neighboring node set v consisting of CNs is denoted as $N(v)$, the symbol $N(v) \setminus c$ denotes the set $N(v)$ except the check node c, $u^0_{v,c}$ is set to 0, add t denotes the number of iteration.

Step 2: From the CNs to VNs the message $u^t_{v,c}$ is passed in the graph as.

$$u^t_{c,v} = max(\min_{v' \in M(c)\backslash v} |u^t_{v',c}| - \beta, 0) \bullet \prod_{v' \in M(c)\backslash v} sign(u^t_{v',c}). \tag{3}$$

where the neighboring node set c consisting of VNs is denoted as $M(c)$, the symbol $M(c) \setminus v$ denotes the set $M(c)$ except the variable node v.

Step 3: The soft estimation message s^t_v after t iterations is written as

$$s^t_v = l_v + \sum_{c' \in N(v)} u^t_{c',v}. \tag{4}$$

Repeat Step 1, 2, 3 until conditions of iterations are reached. The recovered binary vector $\hat{x}^t_v$ is computed by s^t_v as

$$\hat{x}^t_v = \begin{cases} 1, & if \quad s^t_v > 0 \\ 0, & otherwise, \end{cases} \tag{5}$$

In the iterative process, we use the Equation (6) below to replace the Equation (3) in step 2 of CNs layers, and the decoding process is collectively called scale min-sim (SMS) decoder.

$$u^t_{c,v} = (\alpha \times \min_{v' \in M(c)\backslash v} |u^t_{v',c}| - \beta) \bullet \prod_{v' \in M(c)\backslash v} sign(u^t_{v',c}). \tag{6}$$

2.2. Neural Offset Min Sum Network and Neural Scale Min Sum Network

The NOMS algorithm takes the messages transmitted by each edge and subtracts a number of learnable parameters. The task of the network decoder is to use the received vector y to recover the messages $\hat{x}_v$. Input layer, output layer and hidden layers make up the network, as shown in Figure 1. Each of the hidden layer has one CNs layer and one VNs layer. The LLR of the received message is computed by Equation (1).

The CNs layer is given as [21]

$$u^t_{c,v} = \max(\min_{v' \in M(c\backslash v)} |u^t_{v',c}| - \beta^t_{v',c}, 0) \bullet \prod_{v' \in M(c\backslash v)} sign(u^t_{v',c}). \tag{7}$$

where the learning parameter is $\beta^t_{v',c}$.

The VNs layer is given as [21,22]

$$u^t_{v,c} = w^t_{v,c}(l_v + \sum_{c' \in N(v\backslash c)} u^{t-1}_{c',v}). \tag{8}$$

where the learning parameter is $w_{v,c}^{t}$.

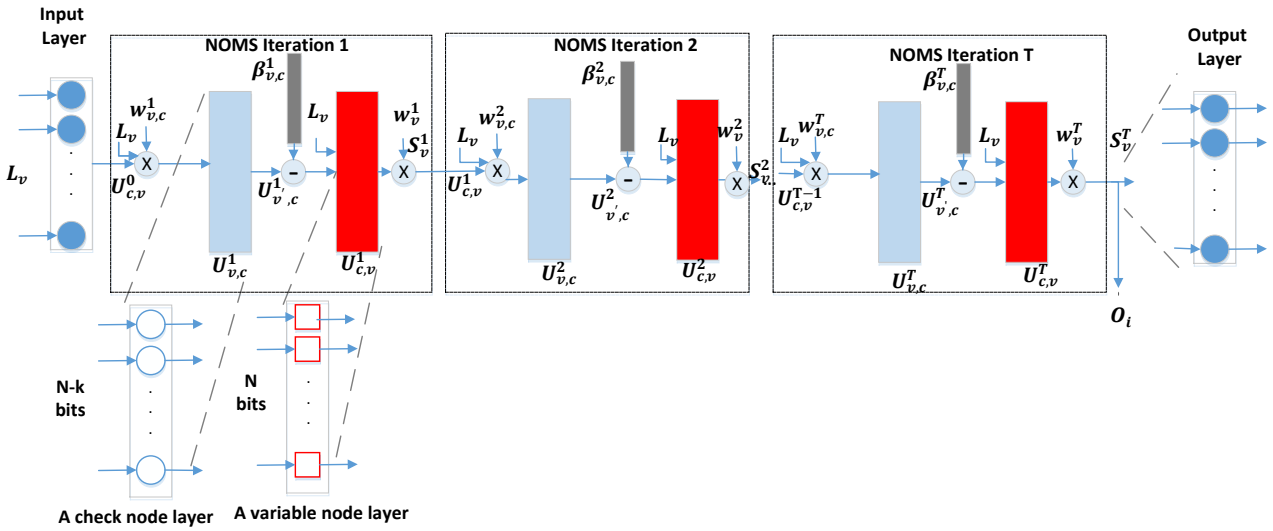


Figure 1. NOMS decoding network.

Finally, the output of each iteration can be expressed as

$$s_v^t = w_v^t(l_v + \sum_{c' \in N(v)} u_{c',v}^t), \tag{9}$$

$$o_i = \sigma(s_v^t) \tag{10}$$

where $\sigma(x) = (1 + e^{-x})^{-1}$ is the activation function. The cross-entropy loss can be expressed as

$$L(x, o) = -\frac{1}{N}\sum_{i=1}^{N} x_i \log(o_i) + (1 - x_i)\log(1 - o_i) \tag{11}$$

where x_i is the information bit transmitted, and o_i is the estimated information bit.

$$\begin{aligned} u_{c,v}^t = \max(\alpha_{v',c}^t \times \min_{v' \in M(c \setminus v)} |u_{v',c}^t| - \beta_{v',c}^t, 0) \\ \bullet \prod_{v' \in M(c \setminus v)} sign(u_{v',c}^t). \end{aligned} \tag{12}$$

In the iterative layer, if we replace Equation (7) with Equation (12) in CNs layers, then it becomes the NSMS decoding algorithm.

2.3. Tensor-Train Decomposition

Assuming that there is a high-order tensor $\mathcal{X}$, a sequence of third-order core tensors $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(N)}$ are used to express $\mathcal{X}$, which is called TT decomposition. $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_n}$ is given by

$$\mathcal{X} = \ll \mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(N)} \gg \tag{13}$$

where $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_n}$, $\mathcal{G} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$, $R_0 = R_N = 1$. The notation $\ll\gg$ represents the approximation of the core tensor sequence to the tensor $\mathcal{X}$. $I_0, I_1, \ldots, I_N$ are used to define the dimension of $\mathcal{X}$. $R_0, R_1, \ldots, R_N$ determine the size of each core tensor, which is called TT rank. Smaller TT rank yields a smaller overall dimension of the series $\mathcal{G}$, which will greatly reduce the complexity. In addition, the $(i_1, i_2, \ldots, i_N)$-th element of the tensor $\mathcal{X}$ can be expressed as

$$x_{i_1, i_2, \ldots, i_N} = \prod_{n=1}^{N} G_{i_n}^{(n)}. \tag{14}$$

where $G_{i_1}^{(1)}, G_{i_2}^{(2)}, \ldots, G_{i_N}^{(N)}$ is the sequence of slices obtained from each core tensor. There is $\mathcal{O}(dIR^2)$ spatial complexity of TT format, where R is the maximum value of the TT ranks, I is the maximum value of modes in $I_1, I_2, \ldots, I_n$, and d is the order of the tensor.

For completeness, the background on TT decomposition is explained in the following. For further studies, readers can refer to [8,23]. TT decomposition employs a sequence of singular value decomposition (SVD) or the cross approximation algorithm. Assume that we want to approximate a given tensor $\mathcal{A}$ by a tensor $\mathcal{X}$, that is $\mathcal{X} \approx \mathcal{A}$, with a prescribed accuracy ϵ. We then need to find the optimal rank and the optimal series of core tensors, $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(N)}$ under the condition $\| \mathcal{X} - \mathcal{A} \|_F \leq \epsilon$. It follows that N consecutive shaping and SVD calculations constitute the computation of the TT decomposition. The detailed steps of TT decomposition via recursive SVD are as follows. First, we validate the TT rank, $R_0, R_1, \ldots, R_N$, with below rules: (1) the length of rank should match with the dimension of $\mathcal{A}$, (2) consecutive factors should have matching ranks, (3) boundary conditions dictate that the head and tail rank set to 1. Second, we set $\mathcal{X}_1 = \mathcal{X}$, and we will find the optimal $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(N-1)}$ one by one. We flatten the unfolding tensor $\mathcal{X}_1$ of the remaining factors, where $\mathcal{X}_1 \in \mathbb{R}^{I_1 \times I}, I = I_2 \times \ldots \times I_n$. We get 1-st TT factor U in $\mathcal{X}_1 = USV^T$ with U as a $I_1 \times I_1$ matrix and V as an $I \times I_1$ matrix. The head TT core $\mathcal{G}^{(1)} = U$ is given by the U indexed by R_0, I_1, R_1. SVD is chosen for its numerical stability in the TT algorithm for binary analysis in this step. Then, the new unfolding tensor for the remaining factors is $\mathcal{X}_1 = SV^T$. Repeat the second step $N-2$ times, and reshape $\mathcal{X}_1$ such that its rows are indexed by $R_{n-1}, I_n, \forall n \in [N-1]$ and its columns by remaining portion. Then we will get each core, $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(N-1)}$ which are reshaped into $\mathcal{G}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$ tensors, $\forall n \in [N-1]$. Finally, the last factor $\mathcal{G}^{(N)} \in \mathbb{R}^{R_{N-1} \times I_N \times R_N}$ is obtained by $\mathcal{X}_1$. Suppose there is a three-dimensional tensor $\mathcal{X} \in \mathbb{R}^{3 \times 4 \times 2}$; we set rank $R_0 = 1, R_1 = 3, R_2 = 2, R_3 = 1$, and the detailed TT decomposition process is shown in Figure 2. By reshaping tensor $\mathcal{X}$ into a 3×8 matrix $\mathcal{X}_1$, SVD of the matrix is then $\mathcal{X}_1 = USV^T$. The head core 3×3 tensors is $\mathcal{G}^{(1)} = U$. Set $\mathcal{X}_1 = SV^T$, $\mathcal{X}_1$ is reshaped into a 12×2 tensor. Again, SVD decomposition was carried out on $\mathcal{X}_1$. The middle and tail core tensors are $\mathcal{G}^{(2)} = U, \mathcal{G}^{(3)} = SV^T$ after reshaping into $3 \times 4 \times 2$ and 2×2. The head and tail core tensors $\mathcal{G}^{(1)}, \mathcal{G}^{(3)}$ are in fact matrices.

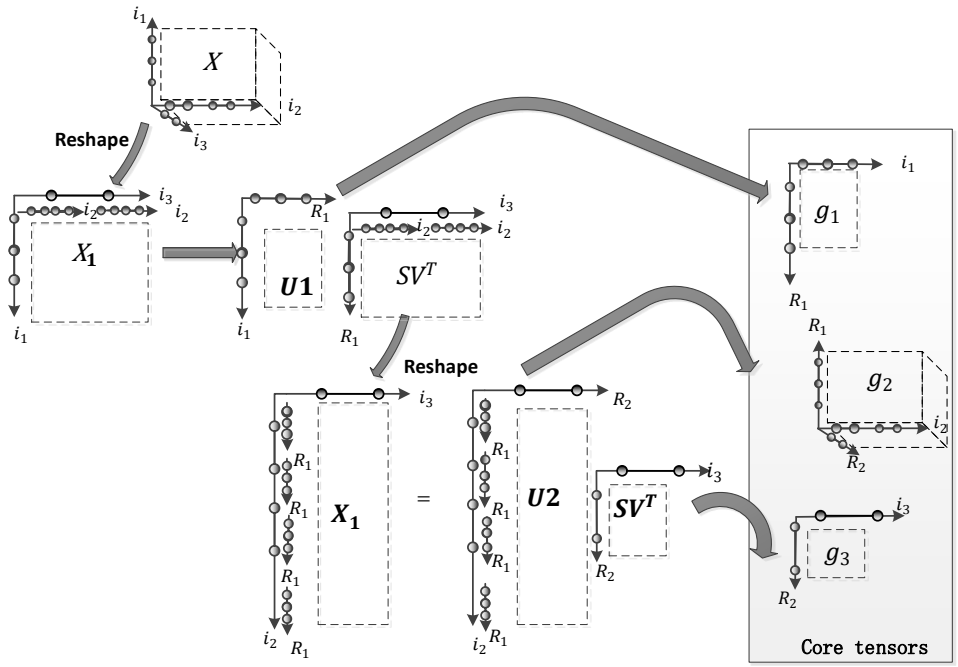


Figure 2. Computation of the TT decomposition.

2.4. Tensor-Ring Decomposition

Tensor ring decomposition is proposed on the basis of tensor train decomposition [16]. The main advantage of the tensor ring decomposition is that the tensor is used in the form of a ring. With the first rank equal to the last rank, so that there are more optimal rank choices. Assuming that $\mathcal{T}$ is a d-dimensional tensor, denoted by $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$. After TR decomposition, $\mathcal{T}$ can be expressed as a series of low rank tensors $\mathcal{Z} \in \mathbb{R}^{r_k \times I_k \times r_{k+1}}$, $k = 1, 2, \ldots, d$, which can be expressed as, according to [16],

$$\mathcal{T}(i_1, i_2, \ldots, i_d) = Tr\{Z_1(i_1)Z_2(i_2)...Z_d(i_d)\}, \tag{15}$$

where $\mathcal{T}(i_1, i_2, \ldots, i_d)$ represents the $(i_1, i_2, \ldots, i_d)$th element of the original tensor $\mathcal{T}$. $Z_k(i_k)$ represents the i_kth slice matrix of the original tensor $\mathcal{T}$. Tensor ring decomposition can also be expressed in the index form:

$$\mathcal{T}(i_1, i_2, \ldots, i_d) = \sum_{a_1, \ldots, a_d = 1}^{r_1, \ldots, r_d} \prod_{k=1}^{d} Z_k(a_k, i_k, a_{k+1}). \tag{16}$$

where k is the index of the tensor dimension, a_k is the k-th rank, and i_k is the index of data dimensions. We can also write Equation (16) in tensor format, given by

$$\mathcal{T} = \sum_{a_1, \ldots, a_d = 1}^{r_1, \ldots, r_d} z_1(a_1, a_2) \circ z_1(a_2, a_3) \circ \ldots \circ z_d(a_d, a_1). \tag{17}$$

where the mathematical notation $\circ$ indicates the outer product of vectors; $z_k(a_k, a_{k+1}) \in \mathbb{R}^{I_k}$ indicates the (a_k, a_{k+1})th mode-2 fiber of tensor $\mathcal{Z}_k$. There is $\mathcal{O}(dIR^2)$ spatial complexity of TR format, which is linear to the order of the tensor d. TR satisfies the property of circular dimensional permutation invariance, which is the property of TR decomposition different from that of TT decomposition.

The steps to decompose the TR tensor in detail are the same as those of TT discussed in Section 2.3. The only difference is that when verifying rank, $R_0 = R_N$ in TR decomposition, So, these tensor cores form a ring. $R_0 = R_N = 1$ in TT decomposition is a special case of TR decomposition, so that TR decomposition subsumes TT decomposition, and its rank selection is more than that of TT decomposition. Thus, there are more chances of obtaining the optimal core sequence tensors $\mathcal{Z} \in \mathbb{R}^{r_k \times I_k \times r_{k+1}}$, $k = 1, 2, \ldots, d$ of optimal rank in TR decomposition.

3. TT-NOMS and TT-NSMS Channel Decoding Algorithm

The NMS algorithm adds a multiplicative factor to the MS algorithm, and then expands it into an NNMS network decoding algorithm through the unfold method. Different from the NMS algorithm, the OMS algorithm adds an additive factor to the MS algorithm, and there is no multiplicative factor. Whether the TT and related tensor decomposition algorithms are suitable for the NOMS algorithm needs further verification. Parameters matrices are decomposed by TT in NOMS algorithm, called TT-NOMS. In Equation (7), the scalar β in the traditional OMS algorithm becomes the tensor $\beta^t_{v',c}$ in each iteration for each edge of NOMS algorithm, which results in additional complexity in the calculation of information transfer for CNs layers. In Equations (8) and (9), tensors $w^t_{v,c}$ and w^t_v are additionally added as training parameters in the neural decoding algorithm. First, we transform the three-dimensional parameter tensors $\beta^t_{v',c}$, $w^t_{v,c}$, w^t_v into four-dimensional tensors $\boldsymbol{\beta}^t_{v',c}$, $\boldsymbol{w}^t_{v,c}$, $\boldsymbol{w}^t_v$ respectively. Then we use TT decomposition on the resulting three four-dimensional tensors as shown below,

$$\boldsymbol{\beta}^t_{v',c} = \beta^{1t}_{v',c}\beta^{2t}_{v',c}\beta^{3t}_{v',c}\beta^{4t}_{v',c}. \tag{18}$$

$$\boldsymbol{w}_{v,c}^{t} = w_{v,c}^{1t} w_{v,c}^{2t} w_{v,c}^{3t} w_{v,c}^{4t}. \tag{19}$$

$$\boldsymbol{w}_{v}^{t} = w_{v}^{1t} w_{v}^{2t} w_{v}^{3t} w_{v}^{4t}. \tag{20}$$

Substituting Equations (18)–(20) into Equations (7)–(9), we have

$$\begin{aligned} u_{c,v}^{t} = \max(\min_{v' \in M(c \backslash v)} |u_{v',c}^{t}| - \beta_{v',c}^{1t} \beta_{v',c}^{2t} \beta_{v',c}^{3t} \beta_{v',c}^{4t}, 0) \\ \bullet \prod_{v' \in M(c \backslash v)} sign(u_{v',c}^{t}), \end{aligned} \tag{21}$$

$$u_{v,c}^{t} = w_{v,c}^{1t} w_{v,c}^{2t} w_{v,c}^{3t} w_{v,c}^{4t} (l_v + \sum_{c' \in N(v \backslash c)} u_{c',v}^{t-1}), \tag{22}$$

$$s_{v}^{t} = w_{v}^{1t} w_{v}^{2t} w_{v}^{3t} w_{v}^{4t} (l_v + \sum_{c' \in N(v)} u_{c',v}^{t}). \tag{23}$$

In the TT-NOMS algorithm, we use Equations (21)–(23) to calculate CNs layers' information, VNs layers' information, and the soft output information, respectively, instead of Equations (7)–(9) in the NOMS algorithm. The loss function is then expressed as

$$L(s^t) = \frac{1}{N-k} \sum_{i=1}^{N-k} max(1 - softsynd(s)_c^t, 0). \tag{24}$$

Syndrome loss function [24] in TT-NOMS is used in Equation (24), where $softsynd(s)_c^t$ denotes the syndrome, t denotes the number of iteration and s_v^t denotes the estimated message. The syndrome loss calculates errors through estimated information s_v^t without the need of labels, so NOMS is called unsupervised neural network. The syndrome can be expressed as

$$softsynd(s)_c^t = \min_{v \in M(c)} s_v^t \prod_{v \in M(c)} sign(s_v^t) \tag{25}$$

In TT-NSMS algorithm, Equation (12) is replaced by Equation (26), two weights $\alpha_{v',c}^{t}$, $\beta_{v',c}^{t}$ and messages $\min_{v' \in M(c \backslash v)} |u_{v',c}^{t}|$ are decomposed by TT into four low-rank weights after reshaping in high order, where $\alpha_{v',c}^{t} = \alpha_{v',c}^{1t} \alpha_{v',c}^{2t} \alpha_{v',c}^{3t} \alpha_{v',c}^{4t}$, $\beta_{v',c}^{t} = \beta_{v',c}^{1t} \beta_{v',c}^{2t} \beta_{v',c}^{3t} \beta_{v',c}^{4t}$, $\min_{v' \in M(c \backslash v)} |u_{v',c}^{t}| = U_{v',c}^{1t} U_{v',c}^{2t} U_{v',c}^{3t} U_{v',c}^{4t}$. That is,

$$\begin{aligned} u_{c,v}^{t} = \max(\alpha_{v',c}^{1t} \alpha_{v',c}^{2t} \alpha_{v',c}^{3t} \alpha_{v',c}^{4t} \times U_{v',c}^{1t} U_{v',c}^{2t} U_{v',c}^{3t} U_{v',c}^{4t} \\ - \beta_{v',c}^{1t} \beta_{v',c}^{2t} \beta_{v',c}^{3t} \beta_{v',c}^{4t}, 0) \bullet \prod_{v' \in M(c \backslash v)} sign(u_{v',c}^{t}). \end{aligned} \tag{26}$$

4. TR-NOMS and TR-NSMS Channel Decoding Algorithm

The TR decomposition has circular dimensional permutation invariance property and is more expressive than the TT decomposition. Better compression ratio and lower storage consumption are obtained in practice. Therefore, we also apply the TR decomposition to the NOMS algorithm, which is called TR-NOMS algorithm. We also transform the three-dimensional parameter tensors $\beta_{v',c}^{t}$, $w_{v,c}^{t}$, w_{v}^{t} into four-dimensional tensors $\boldsymbol{\beta}_{v',c}^{t}$, $\boldsymbol{w}_{v,c}^{t}$,

$\boldsymbol{w}_v^t$ respectively. Then we use TR decomposition on the resulting three four-dimensional tensors as shown below,

$$\boldsymbol{\beta}_{v',c}^{r(t)} = \beta_{v',c}^{r(1t)} \beta_{v',c}^{r(2t)} \beta_{v',c}^{r(3t)} \beta_{v',c}^{r(4t)}. \tag{27}$$

$$\boldsymbol{w}_{v,c}^{r(t)} = w_{v,c}^{r(1t)} w_{v,c}^{r(2t)} w_{v,c}^{r(3t)} w_{v,c}^{r(4t)}. \tag{28}$$

$$\boldsymbol{w}_{v}^{r(t)} = w_{v}^{r(1t)} w_{v}^{r(2t)} w_{v}^{r(3t)} w_{v}^{r(4t)}. \tag{29}$$

Substituting Equations (27)–(29) into Equations (7)–(9), we have

$$\begin{aligned} u_{c,v}^t = \max(\min_{v' \in M(c \backslash v)} |u_{v',c}^t| - \beta_{v',c}^{r(1t)} \beta_{v',c}^{r(2t)} \beta_{v',c}^{r(3t)} \beta_{v',c}^{r(4t)}, 0) \\ \bullet \prod_{v' \in M(c \backslash v)} sign(u_{v',c}^t), \end{aligned} \tag{30}$$

$$u_{v,c}^t = w_{v,c}^{r(1t)} w_{v,c}^{r(2t)} w_{v,c}^{r(3t)} w_{v,c}^{r(4t)} (l_v + \sum_{c' \in N(v \backslash c)} u_{c',v}^{t-1}), \tag{31}$$

$$s_v^t = w_v^{r(1t)} w_v^{r(2t)} w_v^{r(3t)} w_v^{r(4t)} (l_v + \sum_{c' \in N(v)} u_{c',v}^t). \tag{32}$$

In the TR-NOMS algorithm, we use Equations (30)–(32) to calculate CNs layers' information, VNs layers' information, and the soft output information, respectively, instead of Equations (7)–(9) in the NOMS algorithm. The product of four low-rank weights replaces the decomposed original weights.

In the TR-NSMS algorithm, the CNs layer is similar to Equation (26), except that the original weight decomposition algorithm is changed to TR, in which we have options for more ranks.

5. Complexity Analysis

We compare the complexity of NOMS, TT-NOMS, TT-NSMS, TR-NOMS and TR-NSMS in storage complexity and computational complexity. We only analyze the parameters and information from the CNs to the VNs in Tanner graph with other parameters being set to 1.

For all of the parameters with T iterations, that is, $\beta_{v',c}^1, \beta_{v',c}^t, \ldots, \beta_{v',c}^T \in \mathbb{R}^{E \times B}$ in Equation (7), the number of parameters is EBT. Denote E as the edge in graph, T as the number of iterations and B as the size of batch. In order to minimize the parameters and multiplication, we reshape the original tensor shape into high order $x_1, \ldots, x_D$, so $EBT = \prod_{d=1}^D x_d$. Then, we use the product of a series of tensors in TT format $\mathcal{B}(1, 2, \ldots, D)$ to replace the original tensor $[\beta_{v',c}^1, \beta_{v',c}^t, \ldots, \beta_{v',c}^T]$, where $\mathcal{B}(d) \in \mathbb{R}^{R_d - 1 \times x_d \times R_d}$ and rank set is $R_0, R_1, \ldots, R_D, \forall d \in [D]$. So, $[\beta_{v',c}^1, \beta_{v',c}^t, \ldots, \beta_{v',c}^T] = \prod_{d=1}^D \mathcal{B}_d$. The TT-NOMS algorithm reduces the number of parameters of $[\beta_{v',c}^1, \beta_{v',c}^t, \ldots, \beta_{v',c}^T]$ from EBT to $\sum_{d=1}^D x_d R_{d-1} R_d$, where x_d, R_{d-1} and R_d are smaller than E, B and T. The few multiplications in TT-NOMS algorithm come from the multiplication of several tensors $\mathcal{B}(1, 2, \ldots, D)$. TR-NOMS algorithm and TT-NOMS algorithm have the same multiplication and parameters, but with more choices of rank set. In NSMS algorithm, there are two kinds of parameters, multiplicative parameters $\alpha_{v',c}^1, \alpha_{v',c}^t, \ldots, \alpha_{v',c}^T \in \mathbb{R}^{E \times B}$ and offset parameters $\beta_{v',c}^1, \beta_{v',c}^t, \ldots, \beta_{v',c}^T \in \mathbb{R}^{E \times B}$ in Equation (12). The number of parameters is $2EBT$. We reshape two kinds of tensor shape into high-order $x_1, \ldots, x_D$, so $EBT = \prod_{d=1}^D x_d$, and use the product of a series of tensors in TT format $\mathcal{A}(1, 2, \ldots, D)$, $\mathcal{B}(1, 2, \ldots, D)$ to replace original tensor $[\alpha_{v',c}^1, \alpha_{v',c}^t, \ldots, \alpha_{v',c}^T]$, $[\beta_{v',c}^1, \beta_{v',c}^t, \ldots, \beta_{v',c}^T]$, where $\mathcal{A}(d) \in \mathbb{R}^{R_d - 1 \times x_d \times R_d}$,

$\mathcal{B}(d) \in \mathbb{R}^{R_d-1 \times x_d \times R_d}$ and rank set is $R_0, R_1, \ldots, R_D$, $\forall d \in [D]$. So, $[\alpha^1_{v',c}, \alpha^t_{v',c}, \ldots, \alpha^T_{v',c}] = \prod_{d=1}^{D} \mathcal{A}_d$, $[\beta^1_{v',c}, \beta^t_{v',c}, \ldots, \beta^T_{v',c}] = \prod_{d=1}^{D} \mathcal{B}_d$. The TT-NSMS algorithm reduces the number of parameters of $[\alpha^1_{v',c}, \alpha^t_{v',c}, \ldots, \alpha^T_{v',c}]$ and $[\beta^1_{v',c}, \beta^t_{v',c}, \ldots, \beta^T_{v',c}]$ from $2EBT$ to $2\sum_{d=1}^{D} x_d R_{d-1} R_d$, where x_d, R_{d-1} and R_d are smaller than E, B and T. In order to reduce more multiplication, we also need to decompose the information in TT-NSMS algorithm. We reshape message tensor shape into high order $y_1, \ldots, y_D$. We use the product of a series of tensors in TT format $U(1, 2, \ldots, D)$ to replace original message, so $\min_{v' \in M(c \backslash v)} |u^t_{v',c}| = \prod_{d=1}^{D} U_d$, where $U(d) \in \mathbb{R}^{R_d-1 \times y_d \times R_d}$ with rank set $R_0, R_1, \ldots, R_D$. The multiplication in TT-NSMS algorithm comes from the multiplication of several tensors $\mathcal{B}(1, 2, \ldots, D)$ and multiplication of $\mathcal{A}(1, 2, \ldots, D)$ and $U(1, 2, \ldots, D)$. The total number of multiplication is $\sum_{d=1}^{D-1} x_d y_d x_{d+1} y_{d+1} R_{d-1} R_d R_{d+1} T + \sum_{d=1}^{D-1} x_d x_{d+1} R_{d-1} R_d R_{d+1}$. The TR-NSMS algorithm and TT-NSMS algorithm have the same multiplication and parameters, but with more choices of rank set.

6. Simulation Results and Discussions

We investigated the decoding performance of various TT-NOMS, TT-NSMS, TR-NOMS and TR-NSMS for Hamming code, BCH codes and LDPC code. The parameters of the simulation are shown in Table 1. Denote E as the edge in graph, T as the number of iterations and B as the size of batch. E_b/N_0 represents the signal-to-noise ratio. The optimal multiplicative coefficient is set to 0.8 in the normalized min-sum (NMS) algorithm, and the optimal offset coefficient is set to 0.15 in the OMS algorithm.

Table 1. Parameters of simulation.

Parameters	Content
Code	*LDPC, BCH, Hamming*
Min frame error	100
Max frame	100,000
Learning rate	0.01
Batch size	120
Optimizer	*Adam*
E_b/N_0	1–8 (dB)
Number of iterations	5, 15

Because the short loops in the tanner graph affect the performance of decoding, the information transmitted by the CNs layer are unreliable, and the VNs layer transmit reliable information. The performance of these layers does not improve due to the increase in weight. Therefore, the weight $w^t_{v,c}$, w^t_v are set to 1 in Equations (8) and (9), which is similarily done in [3]. $w^t_{v,c}$, w^t_v are not decomposed by TT or TR. All of w in Equations (22), (23), (31) and (32) are set to 1. Tensorly [25] of low rank decomposition is used in our codes.

6.1. TT-NOMS and TT-NSMS

First, the performance of the proposed TT-NOMS, TT-NSMS algorithms with the NOMS [22] and NSMS algorithms are obtained, in which we use five iterations. As shown in Figure 3, when the decoding network has 12 layers in five iterations, the bit error rate (BER) and frame error rate (FER) performances of TT-NOMS, TT-NSMS algorithms and NOMS, NSMS algorithms are basically the same for the Hamming(7,3) code. However, TT-NOMS and TT-NSMS algorithms only use 1.5% the number of parameters in NOMS and NSMS algorithm while the multiplication of TT-NSMS is reduced 222 times, as listed in Table 2. TT-NOMS adds low-rank multiplication to reduce the number of parameters.

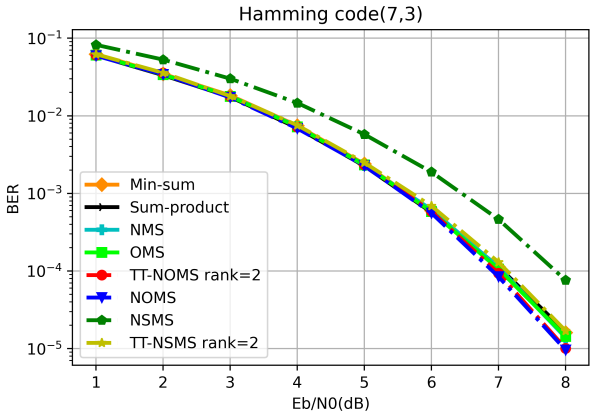


Figure 3. BER performance of TT-NOMS, TT-NSMS, NOMS [22] and NSMS algorithms for Hamming(7,3) code.

Table 2. Comparison of parameters and multiplication in NOMS [22], NSMS, TR-NOMS and TR-NSMS algorithms.

Code	Algorithm	Number of Parameters	Parameter Reduction Percentage	Number of Multiplications
LDPC(576,432)	NOMS	1,224,000	100%	0
	NSMS	2,448,000	100%	146,880,000
	TT-NOMS	474	0.03%	25,780
	TR-NOMS	474	0.03%	25,780
	TT-NSMS	948	0.03%	88,700
	TR-NSMS	948	0.03%	88,700
BCH(127,106)	NOMS	604,800	100%	0
	NSMS	1,209,600	100%	72,576,000
	TT-NOMS	430	0.70%	19,692
	TR-NOMS	430	0.70%	19,692
	TT-NSMS	860	0.70%	67,812
	TR-NSMS	860	0.70%	67,812
BCH(63,45)	NOMS	259,200	100%	0
	NSMS	518,400	100%	31,104,000
	TT-NOMS	316	0.12%	7584
	TR-NOMS	316	0.12%	7584
	TT-NSMS	632	0.12%	21,264
	TR-NSMS	632	0.12%	21,264
Hamming(7,3)	NOMS	7200	100%	0
	NSMS	14,400	100%	864,000
	TT-NOMS	108	1.5%	1008
	TR-NOMS	108	1.5%	1008
	TT-NSMS	216	1.5%	3888
	TR-NSMS	216	1.5%	3888

As shown in Figures 4 and 5, when the decoding network has 12 layers, there is not much difference in BER performance between TT-NOMS, TT-NSMS algorithm, NOMS and NSMS algorithms for BCH(63,45) and BCH(127,106) codes. However, TT-NOMS algorithm only uses 0.12% and 0.7% number of parameters in NOMS algorithm, while at the same time, the multiplication of TT-NSMS is reduced rapidly, as listed in Table 2.

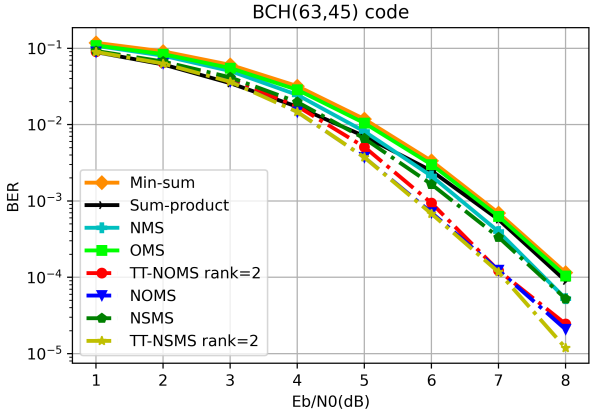


Figure 4. BER performance of TT-NOMS, TT-NSMS, NOMS [22] and NSMS algorithms for BCH(63,45) code.

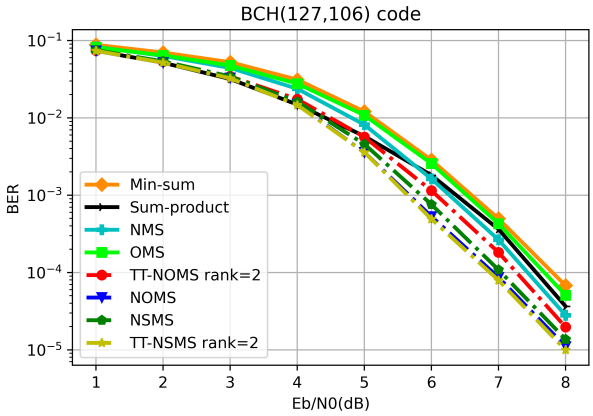


Figure 5. BER performance of TT-NOMS, TT-NSMS, NOMS [22] and NSMS algorithms for BCH(127,106) code.

As shown in Figure 6, the BER performances of the TT-NOMS algorithm and NOMS algorithm are basically the same for LDPC(576,432) code. However, TT-NOMS algorithm only uses 0.03% the number of parameters in NOMS algorithm while the multiplication of TT-NSMS is reduced from 14,688 W to 8.9 W as listed in Table 2.

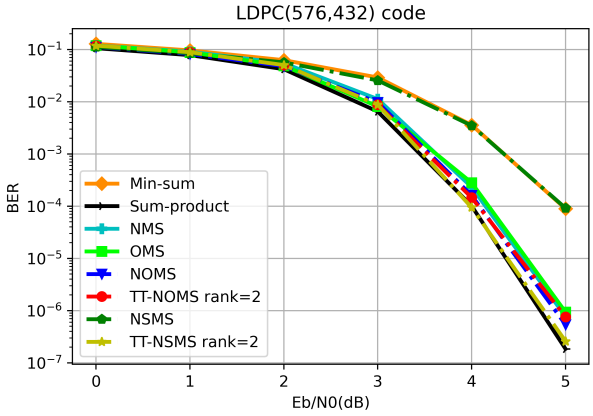


Figure 6. BER performance of TT-NOMS, TT-NSMS, NOMS [22] and NSMS algorithms for LDPC(576,432) code.

6.2. TR-NOMS and TR-NSMS

Figures 7–9 show the performance of the NOMS and TR-NOMS algorithms for hamming and BCH codes. There is not much difference in BER performance between TR-NOMS algorithm, TT-NOMS algorithm, TR-NSMS algorithm and TT-NSMS algorithm. When the two algorithms have the same rank, the reduction of parameters and multiplication are the same.

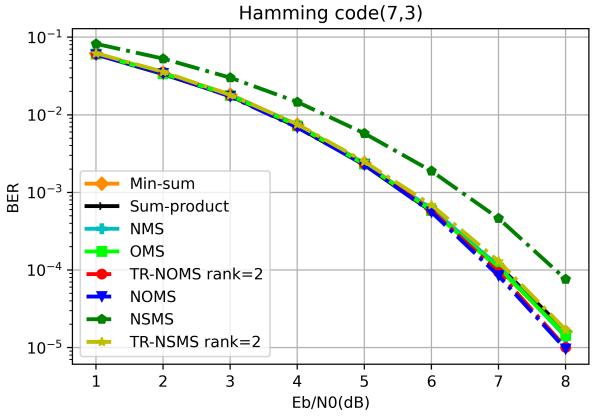


Figure 7. BER performance of TR-NOMS, TR-NSMS, NOMS [22] and NSMS algorithms for Hamming code.

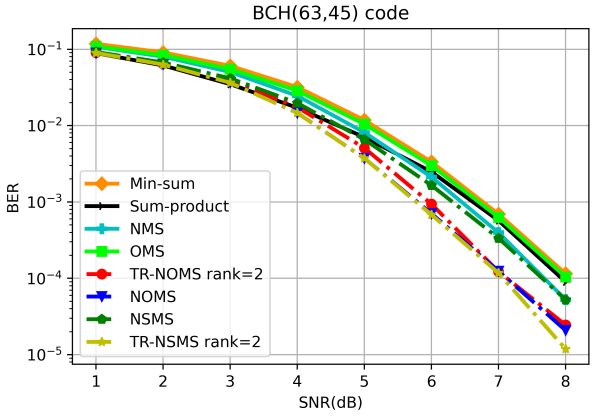


Figure 8. BER performance of TR-NOMS, TR-NSMS, NOMS [22] and NSMS algorithms for BCH(63,45) code.

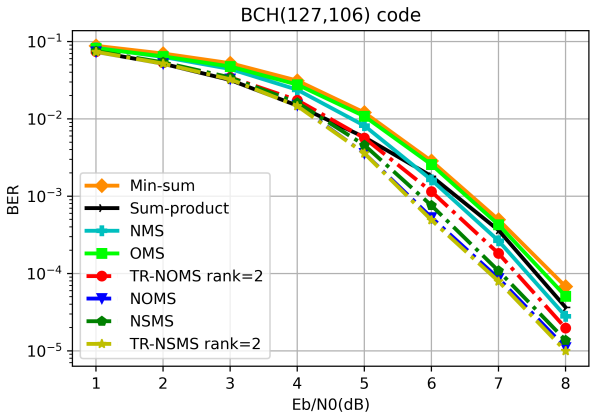


Figure 9. BER performance of TR-NOMS, TR-NSMS, NOMS [22] and NSMS algorithms for BCH(127,106) code.

As shown in Figure 10, performance in TR-NOMS algorithm and NOMS algorithm are basically the same for LDPC(576,432) code. However, TR-NOMS algorithm only uses 0.03% the number of parameters in NOMS algorithm and the same saving is achieved by TT-NOMS for the two algorithms have the same rank while the multiplication of TR-NSMS is reduced rapidly, as listed in Table 2. Figure 11 shows algorithms with 5 and 15 iterations, in which the decomposed network has continuous stability. Figure 12 shows TR-NOMS algorithm for LDPC with different ranks. As the rank increases, the performance improves to a certain extent but the number of parameters and multiplication increase. We can choose different ranks according to our needs. The choice of the optimal rank is another researchable issue.

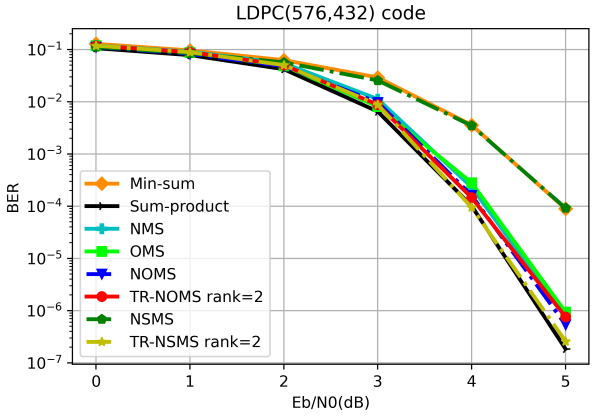


Figure 10. BER performance of TR-NOMS, TR-NSMS, NOMS [22] and NSMS algorithms for LDPC(576,432) code.

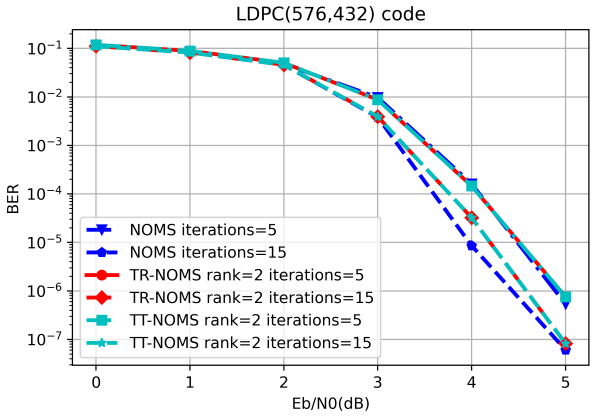


Figure 11. BER performance of TT-NOMS TR-NOMS and NOMS [22] algorithms for LDPC(576,432) code with 5 and 15 iteration.

6.3. Complexity Analysis

The number of parameters and multiplications in NOMS, NSMS, TT-NOMS, TT-NSMS, TR-NOMS and TR-NSMS algorithms are compared in Table 3, where E is the number of edges in Tanner graph, B is the number of batch size, T is the number of iterations. The number of multiplications in Tables 2 and 3 include multiplications in CNs layers.

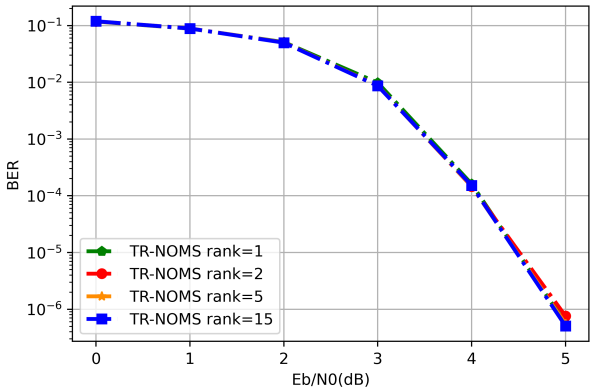


Figure 12. BER performance of TR-NOMS algorithms for LDPC(576,432) code with different rank.

Table 3. Comparison of parameters and multiplication in NOMS [22], NSMS, TT-NOMS, TT-NSMS, TR-NOMS and TT-NSMS algorithms.

Algorithm	Number of Parameters	Number of Multiplications
NOMS	EBT	0
NSMS	$2EBT$	EB^2T
TT-NOMS or TR-NOMS	$\sum_{d=1}^{D} x_d R_{d-1} R_d$	$\sum_{d=1}^{D-1} x_d x_{d+1} R_{d-1} R_d R_{d+1}$
TT-NSMS or TR-NSMS	$2\sum_{d=1}^{D} x_d R_{d-1} R_d$	$\sum_{d=1}^{D-1} x_d y_d x_{d+1} y_{d+1} R_{d-1} R_d R_{d+1} T + \sum_{d=1}^{D-1} x_d x_{d+1} R_{d-1} R_d R_{d+1}$

The same rank is used in TT and TR decomposition. We only use 108 parameters in the TT-NOMS and TR-NOMS algorithms and 216 parameters in TT-NSMS and TR-NSMS algorithms for Hamming(7,3) code, and the percentage reduction of parameters is 1.5%. The number of multiplication has been reduced to 0.45% in TT-NSMS and TR-NSMS algorithms. We only use 316 parameters in the TT-NOMS and TR-NOMS algorithms and 632 parameters in TT-NSMS and TR-NSMS algorithms for BCH(63,45) code, and the percentage reduction of parameters is 0.12%. The number of multiplication has been reduced to 0.0068% in TT-NSMS and TR-NSMS algorithms.We only use 430 parameters in the TT-NOMS and TR-NOMS algorithms and 860 parameters in TT-NSMS and TR-NSMS algorithms in BCH(127,106) code, and the percentage reduction of parameters is 0.70%. The number of multiplication has been reduced to 0.0093% in TT-NSMS and TR-NSMS algorithms. We use 474 parameters in the TT-NOMS and TR-NOMS algorithms and 984 parameters in TT-NSMS and TR-NSMS algorithms for LDPC(576,432) code, the percentage reduction of parameters is 0.03%. The number of multiplication has been reduced to 0.006% in TT-NSMS and TR-NSMS algorithms. TT-NOMS, TR-NOMS, TT-NSMS, TR-NSMS algorithms can reduce the storage space and computational complexity during deployment.

7. Conclusions

This paper investigates low-complexity NOMS and NSMS decoding algorithms using TT decomposition and TR decomposition, which have almost the same performance, but lower computational complexity and fewer parameters in comparison with the the NOMS and NSMS algorithms. We have also provided an analytical framework on the complexity of the TT-NOMS, TT-NSMS, TR-NOMS and TT-NSMS algorithms. We conducted experiments on the proposed TT-NOMS, TT-NSMS, TR-NOMS and TT-NSMS algorithms for BCH codes, Hamming code and LDPC code, and the experimental results show that among these codes, the TT-NOMS, TT-NSMS, TR-NOMS and TT-NSMS decoding algorithms have comparable decoding performances, while the proposed algorithms have fewer parameters and reduced multiplications.

Author Contributions: Conceptualization, Y.L. and Q.W.; Methodology, Y.L.; Validation, Q.W.; Formal analysis, Q.W.; Investigation, Q.W.; Resources, Q.W. and Y.L.; Data curation, Q.W.; Writing—original draft, Q.W. and Y.L.; Writing—review and editing, B.K.N., C.-T.L. and Y.M.; Visualization, Y.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the research funding of the Macao Polytechnic University, Macao SAR, China (Project no. RP/ESCA-02/2021).

Institutional Review Board Statement: The study was conducted in accordance with the Declaration of Helsinki.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: The data that support the findings of this study are available from the corresponding author, upon reasonable request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nachmani, E.; Be'ery, Y.; Burshtein, D. Learning to decode linear codes using deep learning. In Proceedings of the 2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton), Monticello, IL, USA, 27–30 September 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 341–346.
2. Nachmani, E.; Marciano, E.; Burshtein, D.; Be'ery, Y. Rnn decoding of linear block codes. *arXiv* **2017**, arXiv:1702.07560.
3. Nachmani, E.; Marciano, E.; Lugosch, L.; Gross, W.J.; Burshtein, D.; Be'ery, Y. Deep learning methods for improved decoding of linear codes. *IEEE J. Sel. Top. Signal Process.* **2018**, *12*, 119–131. [CrossRef]
4. Qingle, W.; Su-Kit, T.A.N.G.; Liang, Y.; Lam, C.T.; Yan, M. A low complexity model-driven deep learning ldpc decoding algorithm. In Proceedings of the 2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS), Chengdu, China, 23–26 April 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 558–563.
5. Tran-Thi, B.N.; Nguyen-Ly, T.T.; Hong, H.N.; Hoang, T. An improved offset min-sum ldpc decoding algorithm for 5 g new radio. In Proceedings of the 2021 International Symposium on Electrical and Electronics Engineering (ISEE), Ho Chi Minh, Vietnam, 15–16 April 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 106–109.
6. Dai, J.; Tan, K.; Si, Z.; Niu, K.; Chen, M.; Poor, H.V.; Cui, S. Learning to decode protograph ldpc codes. *IEEE J. Sel. Areas Commun.* **2021**, *39*, 1983–1999. [CrossRef]
7. Chen, C.; Xu, Y.; Ju, H.; He, D.; Zhang, W.; Zhang, Y. Variable correction for min-sum ldpc decoding applied in atsc3.0. In Proceedings of the 2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), Valencia, Spain, 6–8 June 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–5.
8. Oseledets, I.V. Tensor-train decomposition. *SIAM J. Sci. Comput.* **2011**, *33*, 2295–2317. [CrossRef]
9. Novikov, A.; Podoprikhin, D.; Osokin, A.; Vetrov, D.P. Tensorizing neural networks. *Adv. Neural Inf. Process. Syst.* **2015**, *28*, 442–450.
10. Zhao, Q.; Sugiyama, M.; Yuan, L.; Cichocki, A. Learning efficient tensor representations with ring-structured networks. In Proceedings of the ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 12–17 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 8608–8612.
11. Huang, H.; Ni, L.; Wang, K.; Wang, Y.; Yu, H. A highly parallel and energy efficient three-dimensional multilayer cmos-rram accelerator for tensorized neural network. *IEEE Trans. Nanotechnol.* **2017**, *17*, 645–656. [CrossRef]
12. Su, J.; Li, J.; Bhattacharjee, B.; Huang, F. Tensorial neural networks: Generalization of neural networks and application to model compression. *arXiv* **2018**, arXiv:1805.10352.
13. Garipov, T.; Podoprikhin, D.; Novikov, A.; Vetrov, D. Ultimate tensorization: Compressing convolutional and fc layers alike. *arXiv* **2016**, arXiv:1611.03214
14. Tjandra, A.; Sakti, S.; Nakamura, S. Compressing recurrent neural network with tensor train. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 4451–4458.
15. Tjandra, A.; Sakti, S.; Nakamura, S.Tensor decomposition for compressing recurrent neural network. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 8–13 July 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–8.
16. Zhao, Q.; Zhou, G.; Xie, S.; Zhang, L.; Cichocki, A. Tensor ring decomposition. *arXiv* **2016**, arXiv:1606.05535.
17. Pan, Y.; Xu, J.; Wang, M.; Ye, J.; Wang, F.; Bai, K.; Xu, Z. Compressing recurrent neural networks with tensor ring for action recognition. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 4683–4690.
18. Wang, W.; Sun, Y.; Eriksson, B.; Wang, W.; Aggarwal, V. Wide compression: Tensor ring nets. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018.
19. Li, X.P.; Liu, Q.; So, H.C. Nonlinear tensor ring network. *arXiv* **2021**, arXiv:2111.06532.

20. Yuanhui, L.; Lam, C.-T.; Ng, B.K. A low-complexity neural normalized min-sum ldpc decoding algorithm using tensor-train decomposition. *IEEE Commun. Lett.* **2022**, *26*, 2914–2918. [CrossRef]
21. Wang, Q.; Wang, S.; Fang, H.; Chen, L.; Chen, L.; Guo, Y. A model-driven deep learning method for normalized min-sum ldpc decoding. In Proceedings of the 2020 IEEE International Conference on Communications Workshops (ICC Workshops), Virtually, 7–11 June 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 1–6.
22. Lugosch, L.; Gross, W.J. Neural offset min-sum decoding. In Proceedings of the 2017 IEEE International Symposium on Information Theory (ISIT), Aachen, Germany, 25–30 June 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1361–1365.
23. Batselier, K.; Liu, H.; Wong, N. A constructive algorithm for decomposing a tensor into a finite sum of orthonormal rank-1 terms. *SIAM J. Matrix Anal. Appl.* **2015**, *36*, 1315–1337. [CrossRef]
24. Lugosch, L.; Gross, W.J. Learning from the syndrome. In Proceedings of the 2018 52nd Asilomar Conference on Signals, Systems, and Computers, Grove, CA, USA, 28–31 October 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 594–598.
25. Kossaifi, J.; Panagakis, Y.; Anandkumar, A.; Pantic, M. Tensorly: Tensor learning in python. *J. Mach. Learn. Res.* **2019**, *20*, 1–6. Available online: http://jmlr.org/papers/v20/18-277.html (accessed on 8 December 2020).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.