REDUCED-COMPLEXITY DEEP NEURAL NETWORK-AIDED CHANNEL CODE DECODER: A CASE STUDY FOR BCH DECODER

Chunhua Deng, Siyu Liao, Bo Yuan

Department of Electrical and Computer Engineering, Rutgers University

ABSTRACT

Error-correcting codes are very important in modern communication systems. In this paper, we investigate efficient reduced-complexity deep neural network (DNN)-aided channel decoders. Specifically, we leverage DNN training to obtain individual scaling parameters for normalized min-sum algorithms, thereby leading to much faster convergence for the same target bit error rate (BER). Also, we propose to compress the DNN-aided channel decoders via weight sharing. A case study on DNN-aided BCH decoders is investigated. Simulation results and hardware complexity analysis show that our method can reduce 2.59 times of memory cost than non-compressed DNN-aided BCH decoders. Meanwhile, compared to the conventional BCH decoders, our method can improve convergence rate by 6 times with similar decoding performance.

Index Terms— Neural Network, Weight Sharing, BCH Decoder

1. INTRODUCTION

Starting from Shannons seminal paper, channel codes have been used as an important error-correcting and channel-protecting technique over the last six decades. By adding well-crafted redundancy to the transmitted messages via proper encoding methods, channel codes can significantly reduce the random or burst errors incurred by the channel noise, and thereby providing strong protection for transmission quality. To date, various types of channel codes, such as BCH codes [1], Turbo codes [2], LDPC codes [3] and Polar codes [4], have been widely adopted and deployed in numerous communication and storage systems, such as wireless communication, deep space communication, solid-state drives and so on.

On the other hand, in the emerging artificial intelligence (AI) era, deep neural networks (DNNs) [5] have become the most popular and important AI technique. Thanks to their strong learning and representation capability, DNNs have achieved unprecedented success in numerous intelligence-demanded tasks, such as object detection, speech recognition, machine translation, etc. Motivated by these successes, today DNNs are deeply reshaping the landscape of various application domains, such as computer vision, natural language processing, robotics, autonomous driving, etc. Moreover, considering the generality and adaptability of DNNs, both academia and industry firmly believe that DNNs will continue to innovate a broad spectrum of applications and become the critical "intelligence enabler" for many traditional science and engineering fields. Specifically, for channel coding, DNNs are already showing their huge potentials and bringing promising innovation for this classical field. Several prior works [6, 7, 8, 9] have observed the error-correcting performance improvement led by DNN-aided decoding, especially for short-length codes. According to their reports, such improvement widely exists for different types of channel codes (such as BCH codes, convolutional codes, polar codes etc.), and can be brought by different types of DNN models (such as fully-connected neural networks, convolutional neural networks, recurrent neural networks etc.)

However, such DNNs-enabled performance improvement is not free. Because DNNs are computation intensive and storage intensive, the DNN-aided channel decoders typically demand much larger computation consumption and memory consumption as compared to their traditional non-DNN counterparts. From the perspective of practical implementations, such increasing cost on computation and storage will greatly impede the deployment of DNN-aided channel decoders in many application scenarios, especially for those resource-constrained energy-constrained embedded applications.

To address this challenge, in this paper, we propose efficient approach to reduce complexity of DNNs-aided channel decoders with retaining similar error-correcting performance. First, we leverage DNN training to obtain individual scaling parameters for normalized min-sum algorithms. Compared to the conventional normalized min-sum algorithms with unified scaling parameter, such DNN-based parameter selecting scheme leads to much faster convergence for the same target bit error rate (BER). Then, we propose to compress the DNN-aided channel decoders via weight sharing. By utilizing efficient clustering technique, the unnecessary redundancy existed in the massive amount of scaling parameters can be removed with negligible performance loss. To validate the proposed approaches, a case study on DNN-aided BCH decoders is investigated. Simulation results and hardware complexity analysis show that our method can reduce 2.59 times of memory cost than non-compressed DNN-aided BCH decoders. Meanwhile, compared to the conventional

BCH decoders, our method can improve convergence rate by 6 times with similar decoding performance.

The rest of this paper is organized as follows. Section 2 briefly reviews the belief-propagation (BP) algorithm that is the underlying decoding algorithm for many modern channel codes. The corresponding min-sum approximation is also reviewed in this section. Section 3 presents the proposed neural scaled min-sum (NSMS) algorithm for performance improvement and weight sharing technique for decoder compression. Experimental validation of the proposed methods on two example BCH codes is conducted in Section 4. Section 5 analyzes the hardware complexity of the example BCH decoders using our methods. Conclusions are drawn in Section 6.

2. BELIEF PROPAGATION ALGORITHM AND MIN-SUM APPROXIMATION

According to coding theory, various types of channel codes, such as BCH codes and LDPC codes, can be decoded over their corresponding bipartite Tanner graph [10] using BP algorithm. Specifically, BP algorithm propagates and updates the probabilistic messages among the variable nodes (VNs) and check nodes (CNs) of Tanner graph. In general, let us denote the message from one variable node v to one check node c and the message from c to v at t-th iteration of BP algorithm as $\mu^t_{c,v}$ and $\mu^t_{v,c}$, respectively. Then, the update of these messages is as follows:

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

and

$$\mu^t_{c,v} = 2 \tanh^{-1} \left(\prod_{v' \in \mathcal{M}(c) \backslash v} \tanh \left(\frac{\mu^t_{v',c}}{2} \right) \right), \tag{2}$$

where l_v is the received log-likelihood ratio (LLR) message for the variable node v, $\mathcal{N}(v)$ is the set of check node c connecting to that specific variable node v, and $\mathcal{M}(c)$ is the set of variable node v connecting to that specific check node c. After several iterations, the final soft output of iteration t is calculated as follows,

$$s^t_v = l_v + \sum_{c' \in \mathcal{N}(v)} \mu^t_{c',v}. \tag{3}$$

Notice that Eqn. 2 is computation expensive due to the existence of hyperbolic tangent function. Hence it is typically approximated by performing the min-sum operation as stated in [11]:

$$\mu^t_{c,v} = \min_{v' \in \mathcal{M}(c) \backslash v} (|\mu^t_{v',c}|) \prod_{v' \in \mathcal{M}(c) \backslash v} sign\left(|\mu^t_{v',c}|\right). \tag{4}$$

Moreover, since the min-sum approximation cause performance loss as compared with standard BP algorithm [12, 13], it is always scaled with a parameter to compensate the approximate error. As indicated in [12, 13], such scaled min-sum (SMS) algorithm usually uses a unified scaling parameter for all the propagated check-to-variable messages.

3. THE PROPOSED APPROACH

3.1. Neural Scaled Min-Sum Algorithm

Although SMS can partially compensate the performance loss, the strategy of using the same scaling parameter for all the check-to-variable messages may not be optimal. This is because such parameter is typically selected by empirical simulation and may not closely approximate original computation. Also, since BP algorithm itself is not optimal over Tanner graph with girth, it is possible that assigning different scaling parameters for different check-to-variable messages may lead to better decoding performance. Such hypothesis, though intuitively, coincides with the observation in [8], where assigning different offsets to different propagated messages can outperform original BP algorithm.

Next, we describe our proposed neural scaled min-sum (NSMS) algorithm. As revealed by its name, the proposed approach uses DNNs to obtain the specific scaling parameter for each check-to-variable message. Similar to prior DNN-aided work [8, 9], the transformation of Tanner graph to DNN structure can be done via unfolding the entire Tanner graph with the required number of iterations. Here each weight of DNN model corresponds to the scaling parameter of one propagated message in one iteration. In that case, the original iterative propagation over two-layer Tanner graph can be represented as the forward propagation over multi-layer DNN.

As indicated in [8, 9], after transforming Tanner graph to DNN, such DNN-aided channel decoder can be trained using the standard backward propagation. Notice that here the loss function is the cross entropy based multi-loss function [9]:

$$\begin{split} L(\mathbf{p}, \mathbf{y}) = -\sum_{i=1,2,\ldots}^{I} \Bigg(\sum_{k=0}^{K-1} \Big((y(k) \log(1 - p(i,k)) + \\ (1 - y(k)) \log p(i,k) \Big) \Bigg), \end{split} \tag{5}$$

where $I, K, \mathbf{y}, \mathbf{p}$ are the number of iterations, message length, expected received bit vector, the network output vector, respectively. The variable $y(k)$ and $p(i,k)$ are the k-th expecting received bit and k-th network output at i-th iteration, respectively. After finishing training, the DNN-aided channel decoders can correct each input received codeword during its inference phase. Notice that for the computation of check-node-layer to variable-node-layer, Eqn. 4 is now revised as:

$$\mu^t_{c,v} = w_{c,v} \min_{v' \in \mathcal{M}(c) \backslash v} (|\mu^t_{v',c}|) \prod_{v' \in \mathcal{M}(c) \backslash v} sign\left(|\mu^t_{v',c}|\right). \tag{6}$$

3.2. Weight Sharing

As mentioned in Section 1, a major disadvantage of DNN-aided channel decoder is the need of computation and storage overhead. Take the proposed NSMS algorithm for example. Because each check-to-variable message has its own assigned scaling parameter, the corresponding memory overhead is huge, thereby further increasing the energy consumption incurred by the increase access to the memory.

To address this challenge, we propose efficient weight sharing (WS) technique to reduce the required number of scaling parameters in DNN-aided channel decoders. Here the key idea is to compress the size of DNN-aided channel decoder since unnecessary redundancy widely exists in the well-trained DNN models [14, 15, 16]. Specifically, we utilize K-means approach [17] to cluster the weights of DNN models of channel decoder, as the scaling parameters of messages, into multiple classes. Then, all the clustered weights in the same class are replaced by the same centroid value of the current class. Consequently, when the number of clusters is small, the required memory cost for storing DNN weights are significantly reduced. In general, if we assume that the trained weights are clustered into S sets, after K-means clustering, each weight $w_{i,j}$ belongs to the s-th cluster, where $0 \leq s < S$, and the centroid value of the s-th set is denoted as centroid(s). Then the clustered weights can be represented as :

$$w_{i,j}^{WS} = centroid(s_{i,j}) \tag{7}$$

where $w_{i,j}^{WS}$ is the final weight that will be stored in the memory. Fig. 1 illustrates the process of weight sharing in the NSMS algorithm. It can be seen that for the example Tanner graph, using weight sharing with 2-class clustering can significantly reduce the number of stored scaling parameters from 9 to 2. In general,the entire NSMS algorithm with weight sharing process is described in Algorithm 1.

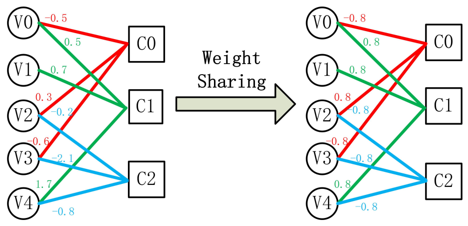

Fig. 1. An example of weight sharing process.

Algorithm 1 NSMS-WS algorithm

```
1: Input : training data, test data
2: Output : Weights $w_{i,j}^{WS}$
3: Build the neural network $f(\mathbf{r},\mathbf{w}) = \sigma(\phi(\mathbf{r},\mathbf{w}))$, where $\mathbf{r}$ is the received symbols, $\mathbf{w}$ is the DNN weights before WS
4: $\mathbf{w} \leftarrow 0$
5: Train the neural network F with loss function (5)
6: End training, getting weights $\mathbf{w}$
7: Using Kmeans to cluster the weights $\mathbf{w}$, getting $s_{i,j}$, $centroid(s)$
8: $w_{i,j}^{WS} \leftarrow centroid(s_{i,j})$
```

4. EXPERIMENTS

To evaluate the performance of the proposed methods, we perform experiments on two types of BCH codes: BCH (63,36) and BCH (63,45). Notice that due to the generality of our approach, it can be applied to any channel codes using BP algorithm.

Experiment Setup: The neural networks that aid for channel coding are trained on TensorFlow platform [18]. The training dataset is composed of 60000 frames of randomly generated data with the signal noise ratio (SNR) equally distributed in the range of [0, 8] dB. The optimizer we used is Adam [19] with a learning rate of 0.001. Regarding the channel condition, all the transmitted codewords are modulated with BPSK and added with additive white Gaussian noise.

Results: Fig. 2 shows the decoding performance of BCH (63,36) and BCH (63,45). From this figure it is seen that the proposed NSMS algorithm can greatly improve the decoding performance as compared to the conventional BP algorithm. In particular, we can discover that NSMS with 5 iterations can achieve the similar BER to the conventional BP algorithm with 30 iterations. Such phenomenon shows that the assigning individual weights for each propagation message can lead to fast decoding convergence. Meanwhile, we also evaluate the effect of different weight sharing schemes by adjusting the required number of bits for representing clustered classes. As shown in Fig. 3, n-bit WS means each clustered weight is replaced by an n-bit centroid, referred as weight index. From this figure it is seen that for the two example BCH codes even using aggressive 1-bit weight sharing scheme can still achieve negligible performance loss. In this case, the memory saving, which will be analyzed in Section 5, is very significant.

5. HARDWARE COMPLEXITY ANALYSIS

In this section, we analyze the hardware complexity of DNN-aided BCH decoder using the proposed approach. Considering the complexity of datapath highly depends on the parallelism, and BP-based channel decoder typically has low degree of parallelism, in this paper we focus on analyzing the memory cost since it is the dominant factor for the entire hardware consumption and energy consumption.

Based on the proposed NSMS algorithm and weight-sharing scheme, we develop the hardware architecture of DNN-aided channel decoder as shown in Fig. 4. In general, it consists of 5 types of SRAMs in the design, including

- Check SRAM: stores checking matrix information.
- WS-idx SRAM: stores weight sharing index information.
- Input SRAM: stores input LLR information.
- V2C SRAM: stores variable-to-check (V2C) messages.

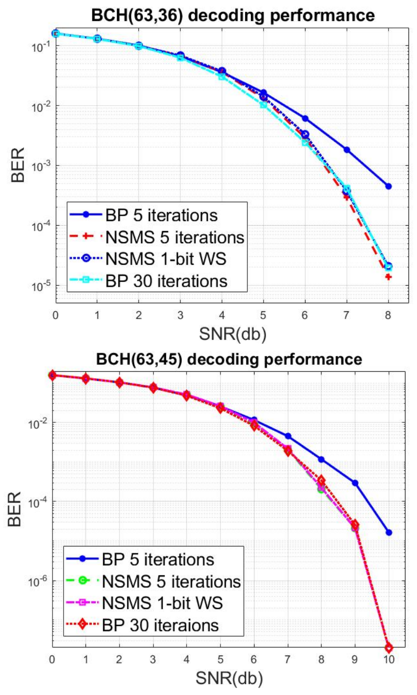

Fig. 2. The decoding performance of BCH (63,36) and BCH (63,45) code.

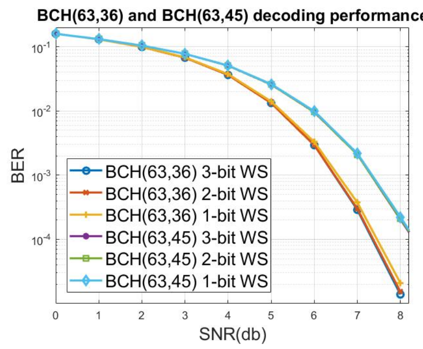

Fig. 3. The performance comparison of BCH (63,36) and BCH (63,45) with different types of weight sharing.

- C2V SRAM: stores check-to-variable (C2V) messages.

At the initialization stage of decoding, the decoderconfigures input SRAM(①), the check SRAM (①), WS-idx SRAM (①) and two common weight registers (①). Meanwhile, the Controller module initializes V2C (②) and C2V (④) SRAMs by filling all elements in these two SRAMs as zero. After that, the decoder begins to start the iteration controlled by the IterCtrl module. At each iteration, the decoder begins with V2C calculation and then C2V calculation. At stage of V2C calculation, the Controller reads data from input SRAM (③) and C2V message (③), reads control information from Check SRAM, and writes the result to V2C SRAM. At stage of C2V calculation, the Controller reads data from V2C message (②), reads control information from Check SRAM (⑥) and from WS-idx SRAM (⑤), and writes the result to C2V SRAM (④). The control information includes the check matrix information and the WS index information.

According to the aforementioned decoding procedure, the hardware system needs to store the message from the check nodes to the variable nodes (C2V) and from the variable nodes to check nodes (V2C) of each edge of Tanner graph. In that case, with typical 8-bit representation for message, as analyzed in Table 1, the total memory requirement of storing the C2V and V2C messages are $16N_e$ bits, where N_e is the number of edges in the Tanner graph. In addition, all of the input LLR should be stored, so the memory requirement for the input SRAM is $8N$ bits, where N is the code length. Then, consider each edge in each iteration needs to store its own weight. Therefore, the total number of additional memory cost after using DNN-aided training is $40N_e$ bits because each edge needs storing 8-bit weight for the whole 5 iterations. On the other hand, after applying our proposed weight sharing scheme, the additional memory requirement is reduced to $5N_e$ bits (each edge decreases from 8-bit to 1-bit) with 1-bit weight sharing. Consequently, the memory saving can be very significant after using weight sharing. For instance, for BCH (63,36) code where $N = 63$, and $N_e = 486$. The NSMS without WS needs 27720 bits memory, while NSMS with WS needs only 10710 bits. Therefore, Weight sharing can reduce the memory cost by 2.59 times with negligible performance loss.

Table 1. Memory cost for BCH (63, 36) decoders.

	BP	NSMS	NSMS with WS
Input	$8N$	$8N$	$8N$
Messages	$16N_e$	$16N_e$	$16N_e$
Weight Index	0	$40N_e$	$5N_e$
Total	$8N$+$16N_e$	$8N$+$56N_e$	$8N$+$21N_e$

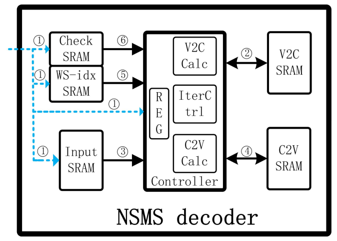

Fig. 4. Hardware architecture of WS-NSMS decoder.

6. CONCLUSION

In this paper, we propose weight sharing-based NSMS algorithm for high-performance reduced-complexity channel decoder. Case study on BCH decoder shows our approach can enable significant saving in iteration time and memory cost.

7. ACKNOWLEDGEMENT

This work is funded by the National Science Foundation Awards CCF-1854737 and CCF-1854742.

8. REFERENCES

[1] Raj Chandra Bose and Dwijendra K Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and control*, vol. 3, no. 1, pp. 68–79, 1960.

[2] Claude Berrou, Alain Glavieux, and Punya Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Communications, 1993. ICC'93 Geneva. Technical Program, Conference Record, IEEE International Conference on*. IEEE, 1993, vol. 2, pp. 1064–1070.

[3] Robert Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[4] Erdal 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.

[5] Geoffrey E Hinton and Ruslan R Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[6] Eliya Nachmani, Yair Be'ery, and David Burshtein, "Learning to decode linear codes using deep learning," in *Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference on*. IEEE, 2016, pp. 341–346.

[7] Tobias Gruber, Sebastian Cammerer, Jakob Hoydis, and Stephan ten Brink, "On deep learning-based channel decoding," in *Information Sciences and Systems (CISS), 2017 51st Annual Conference on*. IEEE, 2017, pp. 1–6.

[8] Loren Lugosch and Warren J Gross, "Neural offset min-sum decoding," in *Information Theory (ISIT), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 1361–1365.

[9] Navneet Agrawal, "Machine intelligence in decoding of forward error correction codes," 2017.

[10] R Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on information theory*, vol. 27, no. 5, pp. 533–547, 1981.

[11] Frank R Kschischang, Brendan J Frey, and H-A Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on information theory*, vol. 47, no. 2, pp. 498–519, 2001.

[12] Bo Yuan and Keshab K Parhi, "Architecture optimizations for bp polar decoders," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 2654–2658.

[13] Bo Yuan and Keshab K Parhi, "Early stopping criteria for energy-efficient low-latency belief-propagation polar code decoders," *IEEE Transactions on Signal Processing*, vol. 62, no. 24, pp. 6496–6506, 2014.

[14] Song Han, Huizi Mao, and William J Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[15] Chunhua Deng, Siyu Liao, Yi Xie, Keshab K. Parhi, Xuehai Qian, and Bo Yuan, "Permdnn: Efficient compressed deep neural network architecture with permuted diagonal matrices," in *Proceedings of the 51th Annual IEEE/ACM International Symposium on Microarchitecture (to be appear)*. ACM, 2018.

[16] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, et al., "C ir cnn: accelerating and compressing deep neural networks using block-circulant weight matrices," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 395–408.

[17] James MacQueen et al., "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Oakland, CA, USA, 1967, vol. 1, pp. 281–297.

[18] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al., "Tensorflow: a system for large-scale machine learning.," in *OSDI*, 2016, vol. 16, pp. 265–283.

[19] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.