FUNCTIONAL ERROR CORRECTION FOR ROBUST NEURAL NETWORKS

A Thesis

by

KUNPING HUANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Anxiao (Andrew) Jiang
Committee Members,	Tie Liu
	Zhangyang (Altas) Wang
Head of Department,	Scott Schaefer

May 2020

Major Subject: Computer Science

ABSTRACT

When neural networks (NeuralNets) are implemented in hardware, their weights need to be stored in memory devices. As noise accumulates in the stored weights, the NeuralNet's performance will degrade. This paper studies how to use error correcting codes (ECCs) to protect the weights. Different from classic error correction in data storage, the optimization objective is to optimize the NeuralNet's performance after error correction, instead of minimizing the Uncorrectable Bit Error Rate in the protected bits. That is, by seeing the NeuralNet as a function of its input, the error correction scheme is function-oriented. A main challenge is that a deep NeuralNet often has millions to hundreds of millions of weights, causing a large redundancy overhead for ECCs, and the relationship between the weights and its NeuralNet's performance can be highly complex. To address the challenge, we propose a Selective Protection (SP) scheme, which chooses only a subset of important bits for ECC protection. To find such bits and achieve an optimized tradeoff between ECC's redundancy and NeuralNet's performance, we present an algorithm based on deep reinforcement learning. Experimental results verify that compared to the natural baseline scheme, the proposed algorithm achieves substantially better performance for the functional error correction task.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Anxiao (Andrew) Jiang (advisor) and Zhangyang (Altas) Wang of the Department of Computer Science and Engineering and Professor Tie Liu of the Department of Electrical and Computer Engineering. It was also supported by Professor Paul Siegel of Department of Electrical and Computer Engineering at University of California, San Diego.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was not supported by any Funding Sources.

NOMENCLATURE

ECC	Error Correcting Code
SP	Selective Protection
BER	Bit Error Rate
UBER	Uncorretable Bit Error Rate
DNN	Deep Neural Network
MSB	Most Significant Bit
LSB	Least Significant Bit
DRL	Deep Reinforcement Learning
BSC	Binary Symmetric Channel
MLP	Multilayer Perceptron
DDPG	Deep Deterministic Policy Gradients
BCH codes	Bose–Chaudhuri–Hocquenghem codes
IEEE	Institute of Electrical and Electronics Engineers

TABLE OF CONTENTS

Page

LIST OF FIGURES

1. INTRODUCTION

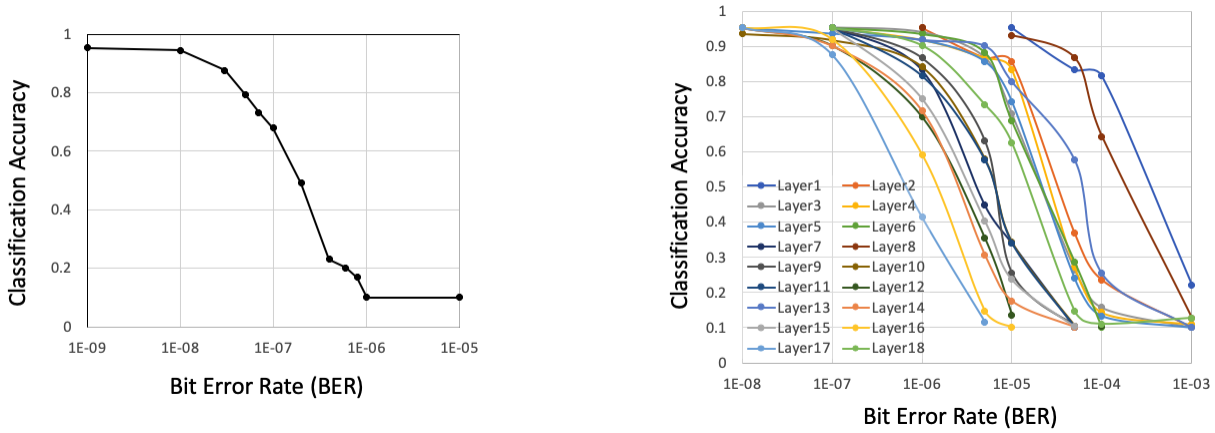
Deep learning has become a boosting force for AI with many applications. When a neural network is implemented in hardware, its weights need to be stored in memory devices. Noise in such devices will accumulate over time, causing the neural network's performance to degrade. It is important to protect neural networks using error correction schemes. In this work, we study how to use error correcting codes (ECCs) to protect the weights of neural networks.

The protection of neural networks has a different optimization objective from classic error correction in data storage systems. In classic error correction, the objective is to minimize the Uncorrectable Bit Error Rate (UBER) in the protected bits. For neural networks, however, the objective is to optimize its performance (e.g., classification accuracy). That is, by seeing the neural network as a function of its input, the error correction scheme is function-oriented.

Several challenges exist for the protection of neural networks. First of all, a deep neural network (DNN) often has many weights. For example, DNNs in computer vision often have millions to hundreds of millions of weights [1]. This can cause a very large redundancy overhead for ECCs. So it is important to design schemes that can reduce redundancy, and achieve an optimized redundancy-performance tradeoff. Such a tradeoff is illustrated in Figure 1.2.

Secondly, the relationship between a neural network's weights and its performance is highly complex. Understanding on the relationship is very limited, and is an active topic of research in many areas [2, 3]. Therefore, it is very challenging to design efficient algorithms that can identify weights that are most important for preserving the performance of neural networks.

We illustrate in Figure 1.1 how a neural network's performance is affected by noise in its weights. The network considered here is ResNet-18 [1], a well-known network for image classification. It consists of 19 layers of nodes and 26 layers of edges (including 8 layers of skip connections). Among the 26 edge layers, 21 of them have trainable weights. When binary-symmetric errors appear in the bits that represent the network's weights, the relation between the Bit Error Rate (BER) and the network's performance (i.e., classification accuracy) is shown in Figure 1.1 (a).



(a) (b)

Figure 1.1: The BER-performance tradeoff for a neural network.

(For a more detailed study on the relation between errors and neural networks' performance, see the nice work in [4].) It can be seen that when the BER is quite small, the network's performance does not degrade much. However, once the BER exceeds a certain threshold, its performance starts to degrade substantially. This relation is common for various types of neural networks [4, 5]. It implies that to protect a neural network, a good redundancy-performance tradeoff can be achieved by keeping the UBER below a certain threshold, especially for those bits that are most critical to the neural network's performance.

We further illustrate that the noise in different layers of a neural network has different impact on its performance. (Similar results have been shown in [4].) We add noise to the weights of only one layer of edges in ResNet-18 at a time, and the result is shown in Figure 1.1 (b). [1] It can be seen that even for the same BER, different layers' noise can impact performance quite differently. Therefore, to optimize the redundancy-performance tradeoff, different layers should receive different levels of protection.

In this paper, we propose a Selective Protection (SP) scheme, which chooses only a subset of important bits for ECC protection. Furthermore, for different layers of edges, the numbers of protected bits for their weights are different. The scheme uses the fact that different layers

[1] For simplicity, the result here is only for the first 18 layers of edges with weights.

impact performance differently. However, since layers jointly determine a network's performance in complex ways, when noise exists in all layers, how to optimize the scheme is still a challenging problem.

To address the challenge, we present an algorithm based on deep reinforcement learning. The key of the algorithm is to learn the complex relation between which bits to protect and the network's corresponding performance. That is, given the knowledge on which bits are protected from errors, we learn a function that can predict the performance of the neural network. We then use the prediction to optimize the set of protected bits, and then the network's corresponding true performance is measured as a feedback reward signal to help further refine the accuracy of the above performance-prediction function. The above learning process repeats itself until its performance converges. To reduce the complexity of learning, we decompose the above process by layers, where the network's layers sequentially take the actions of performance prediction and bit selection. Note that the bits selected for protection in each layer can be a mask vector instead of a single number, that is, we need to decide *which* bits to protect instead of just *how many* bits to protect. That is due to an interesting finding in this paper that, depending on how weights are represented as bits, those bits most worthy of protection are not necessarily the Most Significant Bits (MSBs). Furthermore, since we focus on optimizing the redundancy-performance tradeoff, the ECC redundancy is set as an integrated component in the reward function.

Our algorithm can be evaluated based on the redundancy-performance tradeoff as follows. Let k_{total} denote the total number of bits used to represent the neural network's weights. Let k_{pro} denote the number of bits we protect with ECCs. Let the ECCs be (n, k) linear codes, where n denotes the codeword length and k denotes the number of information bits. Then the number of parity-check bits is $\frac{n-k}{k} \cdot k_{pro}$. We normalize it by k_{total}, and call it *redundancy* r, namely,

$$r = \frac{k_{pro}(n-k)}{k_{total}k}. \tag{1.1}$$

As for the performance of the neural network, for classification tasks (which this work focuses on),

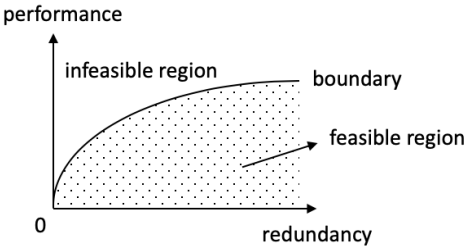


Figure 1.2: The redundancy-performance tradeoff for protecting a neural network.

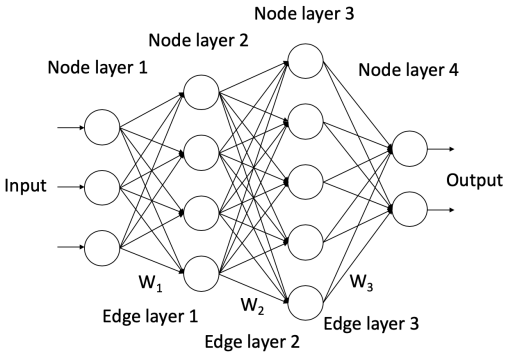


Figure 1.3: A neural network with four node layers (an input layer, two hidden layers and an output layer) and three edge layers. Here W_1, W_2, W_3 are the set of weights in each edge layer.

it usually refers to the classification accuracy, namely, the probability that the inputs are classified correctly.

We compare the performance of our algorithm to a natural baseline scheme, where all layers of the neural network receive the same level of protection from ECCs. Experimental results verify that our proposed algorithm achieves substantially better performance. For example, when the neural network is ResNet-18 and its weights are represented by bits using the IEEE-754 standard (i.e., the single-precision floating-point format), and when BER is 1%, the baseline scheme's classification accuracy drops very quickly once its redundancy r is below the threshold 0.04525. In comparison, our algorithm can decrease the corresponding threshold to 0.03879, which represents a reduction of 14.3% in the redundancy requirement. If the ECC approaches the Shannon capacity, this reduction can be further enlarged to 25.7%.

The rest of the paper is organized as follows. In Section 2, we review related works. In Section 3, we introduce the SP scheme, and present its deep reinforcement learning algorithm. In Section 4, we evaluate the SP scheme by experiments, which verify that the scheme can substantially improve the redundancy-performance tradeoff for neural networks. The results also show that interestingly, depending on how weights are represented as bits, the bits that are most impor-

tant to protect are not necessarily MSBs in the data representation. We present a detailed analysis for this interesting phenomenon. In Section 5, we present concluding remarks.

2. RELATED WORK

The topic explored in this paper is related to several research areas. They include robustness of neural networks against noise, model compression, and reliability of computational circuits.

In the area of *robustness of neural networks against noise*, researchers have studied the effect of noise on the performance of neural networks. In [4], Qin *et al.* studied random bit errors for weights stored as bits, and developed an ECC with one parity bit to improve the network's performance and robustness. In [6], Upadhyaya *et al.* studied random noise for weights stored as analog numbers, and developed analog ECCs to correct the analog noise. In [7, 8], several security attack methods were tested to find specific error patterns that can cause serious damage to neural networks' performance. Note that different from the above works, this paper proposes the Selective Protection scheme for the first time, which protects different sets of bits for different layers. The scheme needs to protect all bits that are critical to the neural network's performance, not just bits that constitute a specific damaging error pattern.

In the area of *model compression*, plenty of works have focused on how to reduce the size of a neural network without affecting its performance [3, 9, 10, 11]. They use various techniques to either prune or quantize the weights in neural networks, and the simplified networks need to be retrained. Deep reinforcement learning methods, including the layer-by-layer training method, have been presented [10, 11]. Note that in our work, we find important bits and protect them, without the need to modify the weights or retrain the network.

In the area of *reliability of computational circuits*, researchers have studied the use of ECCs to ensure the correctness of circuits [12, 13, 14]. In comparison, our work focuses on the redundancy-performance tradeoff, where the neural network's performance does not have to be the same before and after ECC protection.

3. SELECTIVE PROTECTION SCHEME BY DEEP REINFORCEMENT LEARNING

In this section, we present the Selective Protection (SP) scheme for functional error correction. It protects the most important bits in weights by ECC in order to achieve an optimized redundancy-performance tradeoff. We first introduce weight representation for neural networks, and define the Selective Protection scheme. We then present a deep reinforcement learning (DRL) algorithm for the SP scheme.

3.1 Weight Representation in Neural Networks

Neural networks have been used widely in deep learning. An example of a neural network is shown in Figure 1.3, which has four node layers and three edge layers between them. Examples of more complex neural networks, including VGG16 and ResNet-18, are shown in Figure 3.1. (Those two networks are important models for computer vision, and will be used in our experiments.) For ResNet-18, the skip connections between two node layers are also considered an edge layer.

There are different ways to represent weights in neural networks as bits. We introduce two important weight representations below. Both of them will be used in experiments.

3.1.1 Standard Floating-Point Representation

IEEE-754 is an international standard for floating-point representation. We adopt its 32-bit version. Given a weight $w \in \mathbb{R}$, let $B_w^{32} = (b_0, b_1, \cdots, b_{31})$ be its binary representation:

$$w = (-1)^{(b_0)_2} \times 2^{(b_1 b_2 \cdots b_8)_2 - 127} \times (1.b_9 b_{10} \cdots b_{31})_2 \tag{3.1}$$

Here b_0 is the *sign bit*, $b_1 b_2 \cdots b_8$ are the *exponent bits*, and $b_9 b_{10} \cdots b_{31}$ are the *fraction bits*. For example, if $B_w^{32} = (00111100001100000000000000000000)$, then $w = (-1)^{(0)_2} \times 2^{(01111000)_2 - 127} \times (1.01100000000000000000000)_2 = (-1)^0 \times 2^{120-127} \times 1.375 = 0.0107421875$. The IEEE-754 standard can represent values between -2^{127} and 2^{127}.

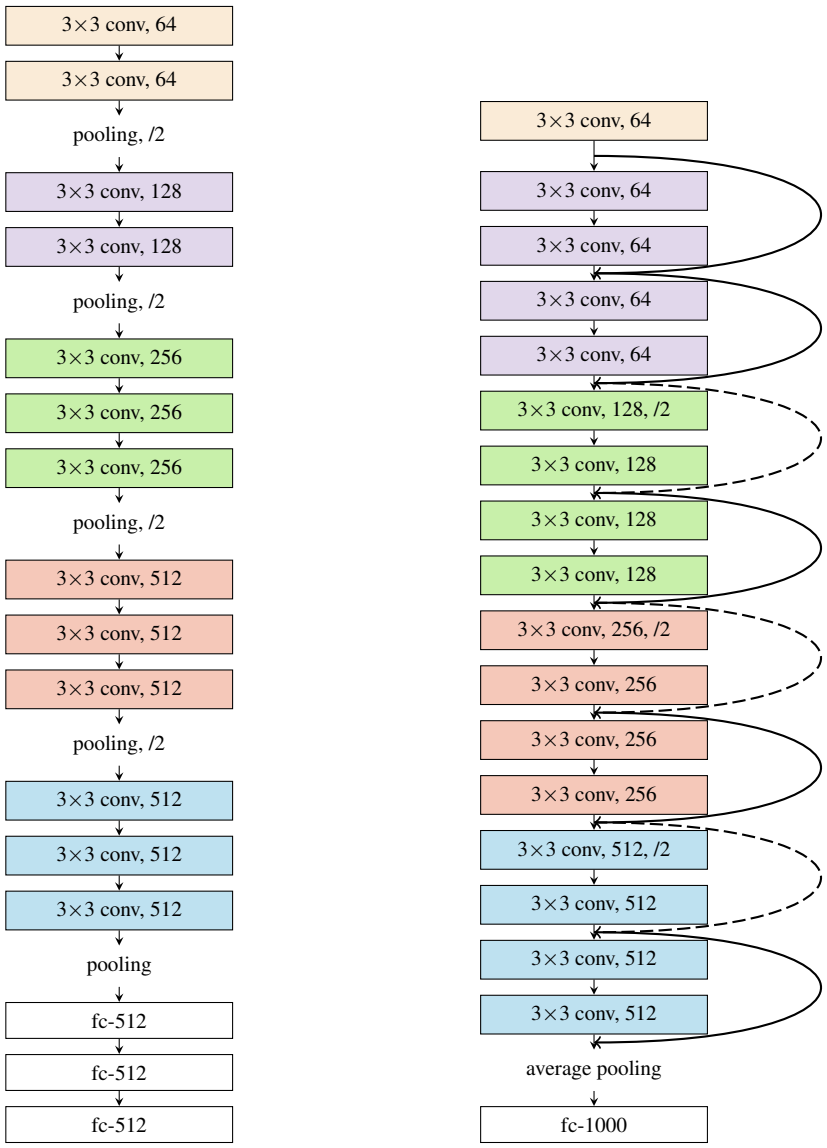


Figure 3.1: The architecture of VGG16 and ResNet-18 models.

3.1.2 Fixed-Point Representation

In this representation, the weights in a range $[-c, c]$ are linearly quantized and represented as bits. (Such a representation has been used in neural networks before, including [11].) Consider its m-bit version. Let $s = c/(2^{m-1} - 1)$ be a scaling factor. Given a weight $w \in [-c, c]$, let $D_w^m = (b_0, b_1, \cdots, b_{m-1})$ be its binary representation:

$$w = (-1)^{(b_0)_2} \times (b_1 b_2 \cdots b_{m-1})_2 \times s \tag{3.2}$$

For example, when $c = 127$ and $m = 8$, if $D_w^m = (10010011)$, then $w = (-1)^{(1)_2} \times (0010011)_2 \times (127/(2^{8-1} - 1)) = (-1)^1 \times 19 \times 1 = -19$.

3.2 Selective Protection Scheme

We now present the Selective Protection (SP) scheme, which selects important bits and protects them from errors with ECCs. Consider a neural network with N edge layers. (In this paper, we consider error protection for weights on edges, not biases in nodes, because biases can often be implemented in alternative ways in hardware. Note that edge weights constitute by far the majority of all weights, and the results here can be naturally extended to biases as well.) For $i = 1, 2, \cdots, N$, let L_i denote the ith edge layer, and let W_i denote the set of weights in L_i. Assume that every weight is represented by m bits. The SP scheme will select a *bit-mask vector*

$$M_i = (\mu_{i,0}, \mu_{i,1}, \cdots, \mu_{i,m-1}) \in \{0, 1\}^m \tag{3.3}$$

for each edge layer L_i. For each weight $w = (b_0, b_1, \cdots, b_{m-1}) \in W_i$, its jth bit b_j will be protected by ECC if $\mu_{i,j} = 1$. Naturally, we let $\mu_{i,j} = 1$ for the layer L_i if its bits in the jth position are critical for the neural network's performance.

Note that the SP scheme applies the same bit-mask vector for all the weights in the same layer. In principle, every weight can be assigned its own bit-mask vector, but that will greatly increase the overhead of the scheme. By using one bit-mask vector per layer, a good balance between

performance and overhead can be achieved.

The neural network has $k_{total} = m\sum_{i=1}^{N}|W_i|$ bits in total. The number of bits protected by ECCs is $k_{pro} = \sum_{i=1}^{N}|W_i|\sum_{j=0}^{m-1}\mu_{i,j}$. When the ECCs are (n,k) linear codes, by Equation (1.1), the *redundancy* of the SP scheme is

$$r(M_1, M_2, \cdots, M_N) = \frac{(n-k)\sum_{i=1}^{N}|W_i|\sum_{j=0}^{m-1}\mu_{i,j}}{km\sum_{i=1}^{N}|W_i|} \tag{3.4}$$

Let $\mathcal{P}(M_1, M_2, \cdots, M_N)$ denote the performance of the neural network (e.g. classification accuracy). Let $\bar{r}$ be a target redundancy. The optimization objective of SP scheme is to maximize $\mathcal{P}(M_1, M_2, \cdots, M_N)$ given that $r(M_1, M_2, \cdots, M_N) = \bar{r}$. That is, after the ECCs are chosen appropriately based on the target Bit Error Rate, the SP scheme can be formulated as

$$\begin{aligned} \max \quad & \mathcal{P}(M_1, M_2, \cdots, M_N) \\ \text{s.t.} \quad & r(M_1, M_2, \cdots, M_N) = \bar{r} \end{aligned} \tag{3.5}$$

3.3 Deep Reinforcement Learning for Selective Protection

We now present a deep reinforcement learning algorithm for the SP scheme. We assume that the bits suffer from errors of a Binary Symmetric Channel (BSC) with Bit Error Rate (BER) p, and a suitable (n,k) linear ECC is used that can correct error of BER p with a probability that approaches 1. Therefore, after error correction, only the bits not protected by ECC will have errors. Note that for a neural network, its performance is a highly complex function of its weights. The DRL algorithm will learn this complex function, and choose the important bits to protect accordingly.

In the following, we first present the essential components of the DRL algorithm: its *state space*, *action space*, *reward function*, and *policy of agents*. We then present the overall learning process of the DRL algorithm.

3.3.1 State Space

There are two types of state spaces in our DRL algorithm: a *Global State Space* and a set of *Local State Spaces*. The global state space uses a set of parameters Θ to characterize the global configuration of the neural network. For $i = 1, 2, \cdots, N$, the ith edge layer has a local state space $\Pi_i \subset \Theta$, which is a partial view of the global state space used by the agent of the ith edge layer to take actions. Note that the parameters in Θ depend on the types of layers in the neural network. In our study, we focus on VGG16 and ResNet, which have two types of layers: convolutional layers and fully-connected layers. Therefore, the parameters in Θ are set accordingly, although they can be adjusted if other types of layers are considered. Note that a fully-connected layer can be seen as a special case of a convolutional layer, where its convolutional kernel has the same size as its input feature map.

For $i = 1, 2, \cdots, N$, let c^i_{in} be the number of input channels for the ith layer L_i (i.e., the number of input feature maps). Let c^i_{out} be its number of output channels (i.e., the number of output feature maps). Let s^i_{kernel} be its kernel size (i.e. the size of its filter for the convolution operation). Let s^i_{stride} be its stride for convolution. Let s^i_{feat} be the size of its input feature map (i.e., each input feature map is a two-dimentional array of size $s^i_{feat} \times s^i_{feat}$). Let $a_i \in \mathcal{A}$ be the most recent action taken by the agent for L_i, where $\mathcal{A}$ denotes the action space, whose details will be introduced later. Let $\alpha_i = (c^i_{in}, c^i_{out}, s^i_{kernel}, s^i_{stride}, s^i_{feat}, |W_i|, a_i)$ denote a state vector associated with L_i. Then, the global state $\theta \in \Theta$ is defined as

$$\theta = (\alpha_1, \alpha_2, \cdots, \alpha_N) \tag{3.6}$$

To simplify the learning process, each layer L_i uses a local state $\pi_i \in \Pi_i$ defined as follows:

$$\pi_i = (c^i_{in}, c^i_{out}, s^i_{kernel}, s^i_{stride}, s^i_{feat}, |W_i|, a_{i-1}) \tag{3.7}$$

When $i = 1$, the parameter $a_{i-1} = a_0$ can be a constant. Note that in π_i, only the action of its

previous layer a_{i-1} is used, instead of the actions of all its previous layers $a_1, a_2, \cdots, a_{i-1}$.

3.3.2 Action Space

We now present the space of actions for the DRL algorithm. For $i = 1, 2, \cdots, N$, the action of the ith layer L_i is to choose a value $a_i \in \{0, 1\}^m$ for its bit-mask vector $M_i = (\mu_{i,0}, \mu_{i,1}, \cdots, \mu_{i,m-1})$. The overall action is the sequence of actions $(a_1, a_2, \cdots, a_N)$. Note that in each iteration of the DRL algorithm, the actions $a_1, a_2, \cdots, a_N$ are chosen sequentially. When the layer L_i takes the action a_i, it chooses the value of a_i (i.e., sets its bit-mask vector M_i) based on its local state π_i and the reward function (to be introduced later).

Let the above method be called the *BitMask* method. To make the method satisfy the redundancy constraint, the reward function not only considers the performance of the neural network, but also the distance between the current redundancy r and the target redundancy $\bar{r}$. The reward value is actually a linear combination of the two terms. When the DRL algorithm ends, the final redundancy r will be close, but not necessarily equal, to $\bar{r}$. By making the coefficient for the distance between r and $\bar{r}$ sufficiently large in the reward value, we can make r sufficiently close to $\bar{r}$.

We now present a simplified version of the *BitMask* method, which we called the *TopBits* method. In the *TopBits* method, each layer always chooses the first few bits of its weights for ECC protection. (The number of bits chosen by different layers can still be different.) This method is intuitively understandable for the fixed-point representation, because the first bit b_0 is the sign bit (thus very important), and for the remaining bits, the More Significant Bits (MSBs) affect the value of the weight more significantly than the Less Significant Bits (LSBs). Similarly, for the IEEE-754 floating-point representation, the first bit b_0 is also the sign bit (thus important), the exponent bits (which follow b_0) affect the weight more significantly than the fraction bits, and the MSBs in the fraction bits affect the weight more significantly than LSBs. Therefore, it seems natural for the SP scheme to always protect the first few bits. The *TopBits* method also simplifies the learning process compared to the *BitMask* method. However, our study will show the surprising result that the *BitMask* method can sometimes outperform the *TopBits* method (namely, MSBs do not always

affect the performance of neural networks more substantially than LSBs).

In the *TopBits* method, the reward function considers only the performance of the neural network, and does not consider the distance between the current redundancy r and the target redundancy $\bar{r}$. To satisfy the redundancy constraint, the method takes two rounds of actions across all the layers in each iteration of the DRL algorithm:

- In the first round, the N layers take actions $(a_1, a_2, \cdots a_N)$ sequentially. For $i = 1, 2, \cdots, N$, the action of the ith layer L_i is to choose a value $a_i \in \{0, 1, \cdots, m\}$, and set the first a_i bits of the bit-mask vector M_i to 1 and set its other bits to 0. Namely, L_i selects the first a_i bits of each weight for ECC protection.

- In the second round, if the current redundancy r is greater than the target redundancy $\bar{r}$, then for $i = 1, 2, \cdots, N$, each layer L_i decreases its a_i by 1 (but without making a_i negative) and adjusts its M_i accordingly. The layers take the above actions sequentially, and stop as soon as we have $r \leq \bar{r}$.

3.3.3 Reward Function

We now present the reward function for the DRL algorithm. Let $\mathcal{P}_0$ describe the performance (e.g., classification accuracy) of the neural network without any bit errors. After each iteration of the DRL algorithm (where the N layers take their actions $(a_1, a_2, \cdots, a_N)$ and set their bit-mask vectors $(M_1, M_2, \cdots, M_N)$ accordingly), random bit errors of BER p are added to all bits in the N layers (but note that some of them are chosen to be protected by ECCs), and then the performance $\mathcal{P}$ of the neural network is measured. For the *TopBits* method, the reward function after the iteration is set as

$$R_{\textit{TopBits}} = \mathcal{P} - \mathcal{P}_0 \tag{3.8}$$

For the *BitMask* method, its reward function also needs to consider the distance between the redundancy r after the iteration and the target redundancy $\bar{r}$. Let β^+ and β^- to be two positive real

numbers. We define a function $f(r, \bar{r})$ as:

$$f(r, \bar{r}) = \begin{cases} \beta^+(\bar{r} - r) & \text{if } r \geq \bar{r} \\ \beta^-(r - \bar{r}) & \text{if } r < \bar{r} \end{cases} \tag{3.9}$$

and define the reward function as:

$$R_{\textit{BitMask}} = \mathcal{P} - \mathcal{P}_0 + f(r, \bar{r}) \tag{3.10}$$

Note that $f(r, \bar{r}) \leq 0$, which represents a penalty for the reward function when the current redundancy r deviates from the target redundancy $\bar{r}$. When $r \geq \bar{r}$ (an undesirable case because the current redundancy is too large), the penalty $\beta^+(\bar{r} - r)$ helps the DRL algorithm reduce the redundancy in the next iteration. When $r < \bar{r}$ (a desirable case because the current redundancy is sufficiently small), interestingly, it is also helpful to set a small penalty $\beta^-(r - \bar{r})$, because it can prevent the neural network from getting stuck in states of very low redundancy in the practical implementation of the DRL algorithm. We usually make β^- much less than β^+. For example, we can set $\beta^+ = 1$ and $\beta^- = 0.05$.

3.3.4 Policy of Agents and the Learning Process

In the DRL algorithm, every layer L_i has an *agent* A_i that takes the action a_i based on the local state π_i and an estimated reward function $\hat{R}$. How the agent A_i chooses the action a_i based on the available information is called its *policy*. In this part, we present the policy of the N agents $A_1, A_2, \cdots, A_N$.

We build four deep neural networks: an *Actor Network*, a *Target Actor Network*, a *Critic Network*, and a *Target Critic Network*. The four networks are illustrated in Figure 3.2. They are all Multilayer Perceptron (MLP) neural networks of four node layers, where the two hidden layers have size 400 and 300, respectively. Additional information on their architectures is as follows:

- *Actor Network* and *Target Actor Network*: For both networks, the input is the local state π_i, and the output is the action a_i. The two networks have similar functions, but update their

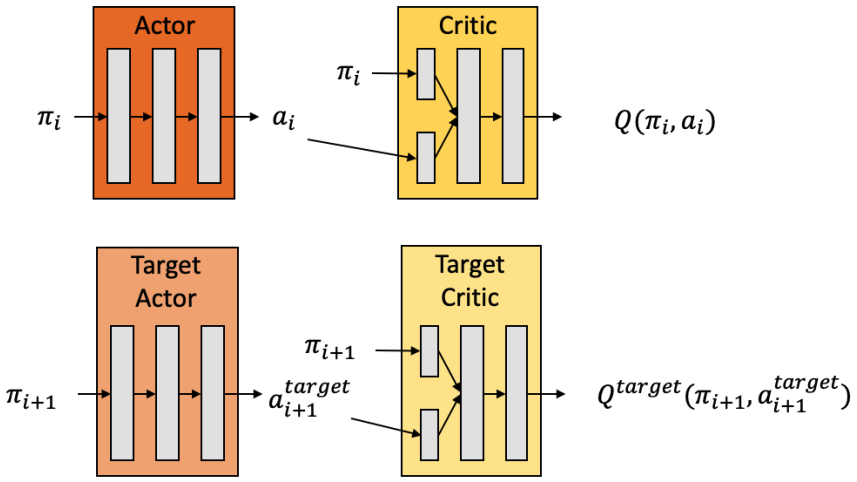


Figure 3.2: The four neural networks used in the deep reinforcement learning algorithm: the Actor Network (top left), the Target Actor Network (bottom left), the Critic Network (top right) and the Target Critic Network (bottom right).

weights with different algorithms during training.

- *Critic Network* and *Target Critic Network*: For both networks, the input consists of the local state π_i and the action a_i, and the output is an estimated value for the summation of the current and the future rewards in the same iteration (where future rewards are discounted in certain ways). Specifically, let γ be a discount factor. Then for $t = 1, 2, \cdots, N$, the output of the two networks is the value of the following Q function:

$$Q(\pi_t, a_t) = \sum_{i=t}^{N} \gamma^{i-t} \hat{R}(\pi_i, a_i) \tag{3.11}$$

  where $\hat{R}(\pi_i, a_i)$ is an estimation of the real reward of this iteration. As before, the two networks also have similar functions, but update their weights differently during training.

The DRL algorithm keeps using the Actor Network to generate actions. In each iteration, the N agents $A_1, A_2, \cdots, A_N$ generate the actions $a_1, a_2, \cdots, a_N$ sequentially. That is, for $i = 1, 2, \cdots, N$, the Actor Network takes π_i as input, and outputs the action a_i. (Note that the Ac-

tor Network outputs real numbers, and we round them to the nearest integers to get the action a_i.) After an iteration, the N local states $(\pi_1, \pi_2, \cdots, \pi_N)$, the N actions $(a_1, a_2, \cdots, a_N)$ and the overall reward R of the iteration are stored in a buffer. The buffer has a fixed size. When new data come in, if the buffer is full, the oldest data will be removed. Therefore, the buffer always stores the most recent results.

After each iteration, a number of samples will be randomly chosen from the buffer to train the four networks. Each sample has the form of $(\pi_i, a_i, \pi_{i+1}, R)$. The four networks update their weights as follows, using the idea of the DDPG algorithm [15]:

- Step 1: *train the Critic Network*. As shown in Figure 3.2, the Critic Network takes π_i and a_i as input, and outputs a value $Q(\pi_i, a_i)$. We also concatenate the Target Actor Network and the Target Critic Network (as shown in Figure 3.2), and use π_{i+1} as input to generate the output $Q^{target}(\pi_{i+1}, a_{i+1}^{target})$. The loss function of the Critic Network is then set as

$$\mathcal{L}_{critic} = (Q(\pi_i, a_i) - \gamma Q^{target}(\pi_{i+1}, a_{i+1}^{target}) - (R - \mathcal{B}))^2 \tag{3.12}$$

  where the baseline $\mathcal{B}$ is defined as an exponential moving average of all previous rewards in order to reduce the variance of gradient estimation. A small number of samples are used as a mini-batch, and their total loss is used to update the weights of the Critic Network via backpropagation.

- Step 2: *train the Actor Network*. We concatenate the Actor Network and the Critic Network (as shown in Figure 3.2), and use π_i to generate the output $Q(\pi_i, a_i)$. The loss function is then set as

$$\mathcal{L}_{actor} = -Q(\pi_i, a_i) \tag{3.13}$$

  Then the total loss of a mini-batch of such samples is used to update the weights of the Actor Network via backpropagation (with the weights of the Critic Network frozen).

- Step 3: *train the Target Actor Network*. Let δ be a small number, such as $\delta = 0.01$. Let

w_{actor}^{target} be a weight of the current Target Actor Network, and let w_{actor} be the corresponding weight of the updated Actor Network. We update w_{actor}^{target} as:

$$w_{actor}^{target} \leftarrow w_{actor}^{target} + \delta(w_{actor} - w_{actor}^{target}) \tag{3.14}$$

We update all weights of the Target Actor Network in the same way.

- Step 4: *train the Target Critic Network*. We update its weights in the same way as we did with the Target Actor Network, except that here we consider the Target Critic Network and the Critic Network.

In summary, the Critic Network learns to predict the future rewards given the current state and the action to be taken. The Actor Network learns to take the best action based on the future rewards predicted by the Critic Network. The Target Critic Network (respectively, the Target Actor Network) follows the learning of the Critic Network (respectively, the Actor Network), except that it updates its weights at a slower pace, which is a conservative method that helps the DRL algorithm converge. The DRL algorithm ends when the four networks' performance converges or when a preset number of training steps is reached.

4. EXPERIMENTAL EVALUATION AND ANALYSIS

In this section, we present experimental evaluation of the Selected Protection scheme. We focus on two important deep neural networks in computer vision: ResNet-18 [1] and VGG16 [16]. We consider two well-known datasets: the CIFAR-10 dataset [17] and the MNIST dataset [18]. We use two data representation schemes for the weights: the IEEE-754 floating-point representation, and the fixed-point representation. We explore two types of error correcting codes: an ideal ECC that reaches the Shannon capacity, and a practical finite-length BCH code. And we study the performance of two methods for the SP scheme: the *BitMask* method and the *TopBits* method.

The experimental results show that the Selective Protection scheme based on deep reinforcement learning can substantially outperform the natural baseline scheme, where all layers protect the same number of bits. The experimental results also reveal a very interesting fact: the Most Significant Bits (MSBs) in a data representation do not always affect the performance of a neural network in the most significant ways. Consequently, the *BitMask* method can sometimes protect some less significant bits (instead of MSBs) and outperform the *TopBits* method. We present a detailed analysis of this surprising finding.

In the following, we introduce the setup of experiments, and present the redundancy-performance tradeoff of the SP scheme. We then show how the *BitMask* method and the *TopBits* method select bits for protection, and analyse why sometimes LSBs are more important for the performance of neural networks than MSBs in noisy environments.

4.1 Setup of Experiments

We test the performance of the SP scheme on two important neural network models: ResNet-18 and VGG16. Both models are commonly used for classifying images, and have various applications in computer vision. The architectures of the two models are illustrated in Figure 3.1. The ResNet-18 network has 26 edge layers and 11.69 million weights. The VGG16 network has 16 edge layers and 138 million weights. Such sizes are typical for deep neural networks.

We perform image classification tasks on two important datasets: the CIFAR-10 dataset and the MNIST dataset. The CIFAR-10 dataset consists of 60, 000 colored images of size 32×32 each, which belong to 10 different classes. The MNIST dataset consists of 70, 000 gray-scaled images of size 28×28 each, which represent the 10 classes of hand-written digits from 0 to 9. Both datasets are widely used for testing the performance of image classification.

We study the SP scheme for two data representation methods: the IEEE-754 floating-point representation and the fixed-point representation. The IEEE-754 representation is an international standard widely used in most hardware systems. The fixed-point representation is a natural alternative way to quantize weights with easily controllable ranges and quantization precision. In our experiments, we let the IEEE-754 representation use 32 bits for each weight, and let the fixed-point representation use 8 bits for each weight.

We explore two types of ECCs for protecting the important bits selected by the SP scheme. The first one is an ideal ECC that reaches the Shannon capacity. When the weights suffer from errors of a binary symmetric channel with BER p, we let the ideal ECC have a code rate of $1 - H(p)$, matching the channel's capacity. We use the code to protect all the selected important bits, and assume that decoding always succeeds. The second type of codes are practical finite-length BCH codes. When the IEEE-754 floating-point representation is used, we let the code be a (8191, 6722) BCH code, which can correct 115 errors. When the fixed-point representation is used, we let the code be a (8191, 6787) BCH code, which can correct 110 errors. When $p = 0.01$ (a practical BER for storage systems), both codes can decode with sufficiently small failure probabilities, thus causing minimal degradation for the neural network's performance.

We study the performance of two methods for the SP scheme: the *BitMask* method and the *TopBits* method. The *BitMask* method offers greater freedom in selecting which bits to protect, while the *TopBits* method offers higher efficiency for learning due to its more restricted solution space. For both methods, the deep reinforcement learning algorithm converges efficiently. Given a solution of the SP scheme, we generate random errors 100 times for all the weights, and evaluate the neural network's average performance (i.e. classification accuracy). The performance was

found to be stable over different experiments.

4.2 Redundancy-Performance Tradeoff

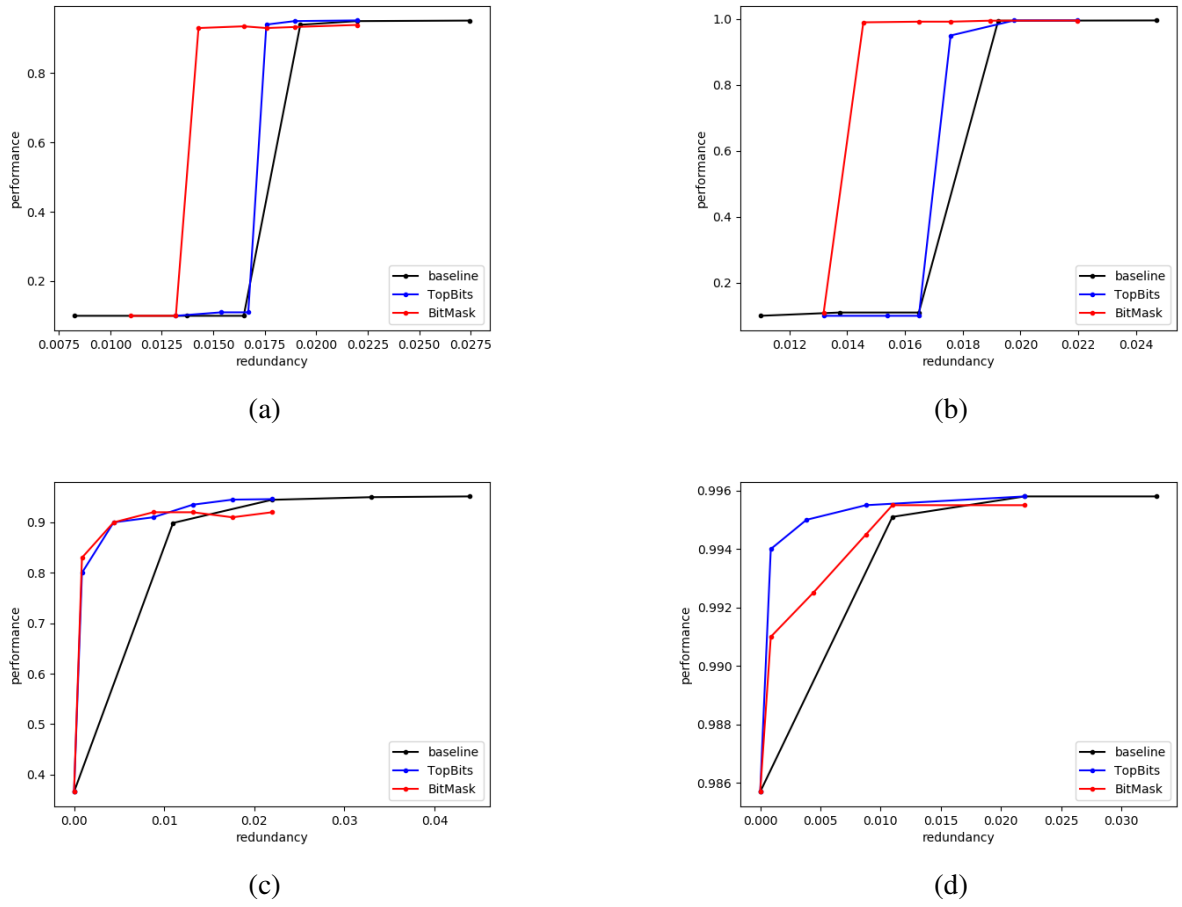


Figure 4.1: The redundancy-performance tradeoff for the SP scheme when ideal ECC is used.

The experimental results for the *redundancy-performance tradeoff* are shown in Figure 4.1 and Figure 4.2. They are for two different types of ECCs, respectively: Figure 4.1 is for the ideal ECC, while Figure 4.2 is for the finite-length BCH codes. In all experiments, we let BER be $p = 0.01$. The redundancy $r = \frac{k_{pro}(n-k)}{k_{total}k}$ can be adjusted by setting different target redundancy in the deep reinforcement learning algorithm. The performance is measured as the average classification accuracy of the neural network, whose noisy weights are partially protected by the ECC.

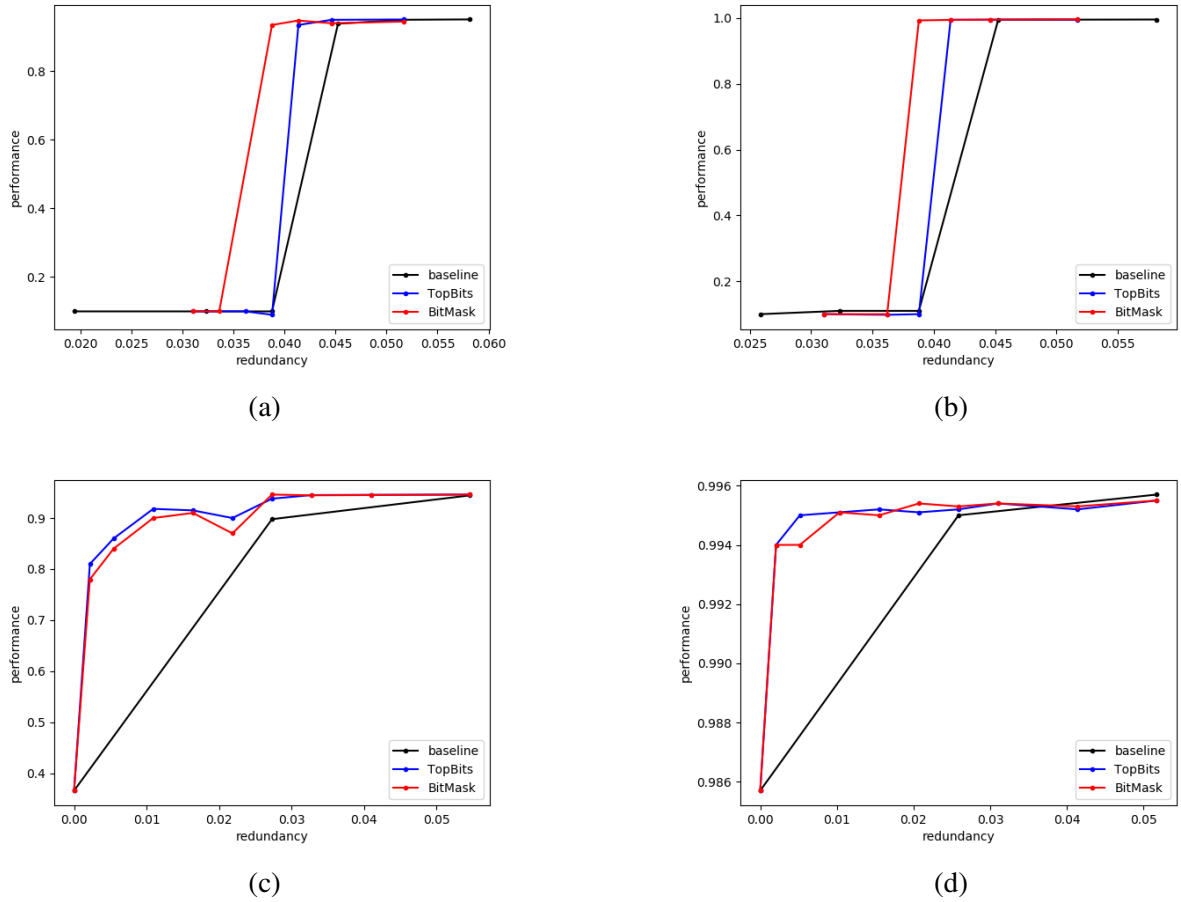


Figure 4.2: The redundancy-performance tradeoff for the SP scheme when BCH codes are used.

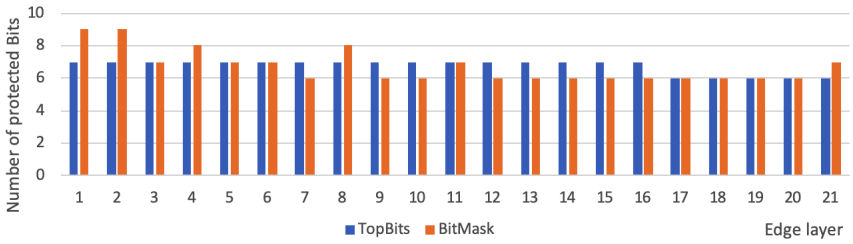
The figures show that when the redundancy r is relatively large, the neural network retains its high performance (because the bits most important for its performance are protected by ECCs). However, once the redundancy drops below a certain threshold, the performance drops sharply. It can be seen clearly that, overall, both the *BitMask* method and the *TopBits* method *significantly* outperform the baseline method, where all layers protect the same number of bits. (In the baseline method, we always protect the first few bits in the weights because they are more significant.)

It can also be seen that when the IEEE-754 representation is used, the *BitMask* method outperforms the *TopBits* method substantially overall. When the fixed-point representation is used, the performance of two methods becomes more comparable, with the *TopBits* method sometimes outperforming the *BitMask* method. It is a very interesting observation because the *TopBits* method

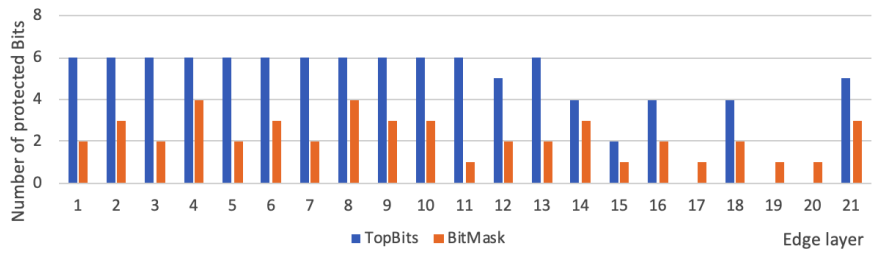
always chooses the first few bits of each weight, which are usually considered more significant than the remaining bits. Furthermore, this restriction also reduces the dimensions of the solution space substantially, which helps improve the efficiency of learning. It implies that the *BitMask* method can find less significant bits that are more important than MSBs for a neural network's overall performance. In the following, we analyse this surprising result by studying how the two methods select bits, and how the bits affect the neural network's performance.

4.3 Bits Protected by Selective Protection Scheme

We now study how the *BitMask* method and the *TopBits* method select bits. For the *number* of bits selected by the two methods, its distribution over the layers is as illustrated in Figure 4.3. It can be seen that when the data representation is IEEE-754, both methods have a relatively even distribution over the layers. And when the data representation is the fixed-point representation, the distribution for both methods becomes less even. Overall, the two methods behave similarly in this aspect.



(a) IEEE-754 floating-point representation



(b) Fixed-point representation

Figure 4.3: The number of selected bits for ECC protection in each edge layer. Here the neural network is ResNet-18, the dataset is CIFAR-10, and the ECC is the ideal ECC.

The major difference between the *BitMask* method and the *TopBits* method is in *which* bits they select. Since their redundancy-performance tradeoff differs most significantly when the IEEE-754 representation is used, we focus on the IEEE-754 representation from now on. For the *TopBits* method, it always selects the first few bits in each layer. For the *BitMask* method, however, it selects bits quite differently. Some typical examples are shown in Figure 4.4. It shows that instead of selecting some more significant bits (such as the third and the fourth bits), the *BitMask* method selects some less significant bits (such as the fifth, sixth and seventh bits in the 11th layer, the 12th layer, $\cdots$, the 16th layer). The result is intriguing because the more significant bits affect the value of a weight more substantially, and are usually expected to affect the performance of the neural network more as well. We present the analysis for the result in the next section.

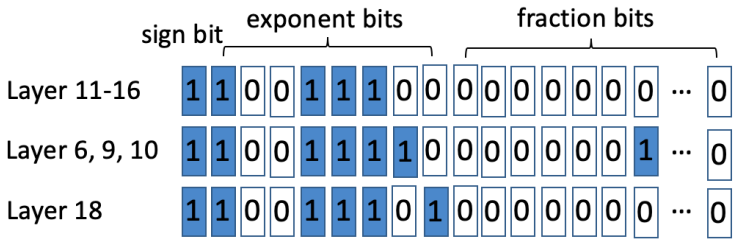


Figure 4.4: Typical examples of the bit-mask vector in some edge layers, with the IEEE-754 floating-point representation and the *BitMask* method.

4.4 Analysis of BitMask Method and TopBits Method

When the IEEE-754 data representation is used, consider a bit among the exponent bits. (The exponent bits are where the *BitMask* method's selection and the *TopBits* method's selection differ the most. For fraction bits, most of them are not selected by either method.) Recall that for a weight, when its bits are $(b_0, b_1, \cdots, b_{31})$, the corresponding weight is $w = (-1)^{(b_0)_2} \times 2^{(b_1 b_2 \cdots b_8)_2 - 127} \times (1.b_9 b_{10} \cdots b_{31})_2$. Let $1 \leq i \leq 8$, and consider the exponent bit b_i. There are two important factors that determine how an error in b_i affects the neural network's performance:

1. Factor one: The 0-to-1 error and the 1-to-0 error have an asymmetric impact on the neural network's performance.

2. Factor two: The bit b_i can have a highly imbalanced probability distribution, which also affects the performance.

We analyze the two factors in the following. For the first factor, consider a 0-to-1 error that changes bit b_i from 0 to 1. In this case, the weight changes from w to $w_{0-to-1} = 2^{2^{8-i}} \times w$. With a 1-to-0 error that changes the bit b_i from 1 to 0, the weight will change from w to $w_{1-to-0} = 2^{-2^{8-i}} \times w$. Since each neuron takes a linear combination of its incoming values before passing it to an activation function, the absolute value of the weight plays an important role in the function of the neuron. It is easy to see that the $0 - to - 1$ error changes the absolute value of the weight much more significantly than the $1 - to - 0$ error. So the $0 - to - 1$ errors are expected to affect the neural network's performance more significantly as well.

We experimentally verify the above observation in Figure 4.6 (a) and (b). They show that when 0-to-1 errors are added, the performance of the neural network drops very sharply. When 1-to-0 errors are added, however, the performance of the neural network does not change much. The results verify that 0-to-1 errors have a more significant impact on the neural network's performance. So to achieve an optimal redundancy-performance tradeoff, there is a strong motivation to protect bits that are more likely to be 0s.

Let us now study the probability distribution of the bits in each bit position. The results are as illustrated in Figure 4.5. It can be seen that for many exponent bits (including bit 1 to bit 6), the probability distribution can be quite uneven. In fact, due to the weight distribution in the neural network, bit 2 and bit 3 here are nearly always 1s, and that explains why they were not selected by the *BitMask* method (as shown in Figure 4.4). Overall, whether a bit should be selected depends on the balance between both factors: the level of asymmetry in the impact on performance by the 0-to-1 errors and the 1-to-0 errors, and the probability for the bit to be 0 or 1. The greater the level of asymmetry is, and the more probable the bit is 0, the more likely the bit will be selected.

We study the bits that are selected differently by the *BitMask* method and the *TopBits* method,

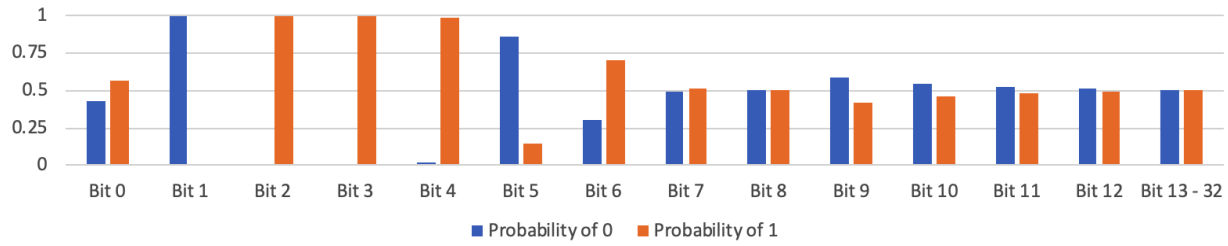


Figure 4.5: The probability distribution of the bits in each bit position.

and explore their impact on the neural network's performance. The experimental results are shown in Figure 4.6 (c) and (d). Let $S_{TopBits}$ be the set of bits selected by the *TopBits* method, and let $S_{BitMask}$ be the set of bits selected by the *BitMask* method. (Here we let the *TopBits* method select the same number of bits as the *BitMask* method in each layer for fair comparison.) It can be seen that when errors are added to the bits in $S_{BitMask} - S_{TopBits}$, the performance of the neural network drops very sharply. When errors are added to the bits in $S_{TopBits} - S_{BitMask}$, however, the performance does not change much. The results verify that the *BitMask* method indeed chooses bits that are more important for the redundancy-performance tradeoff.

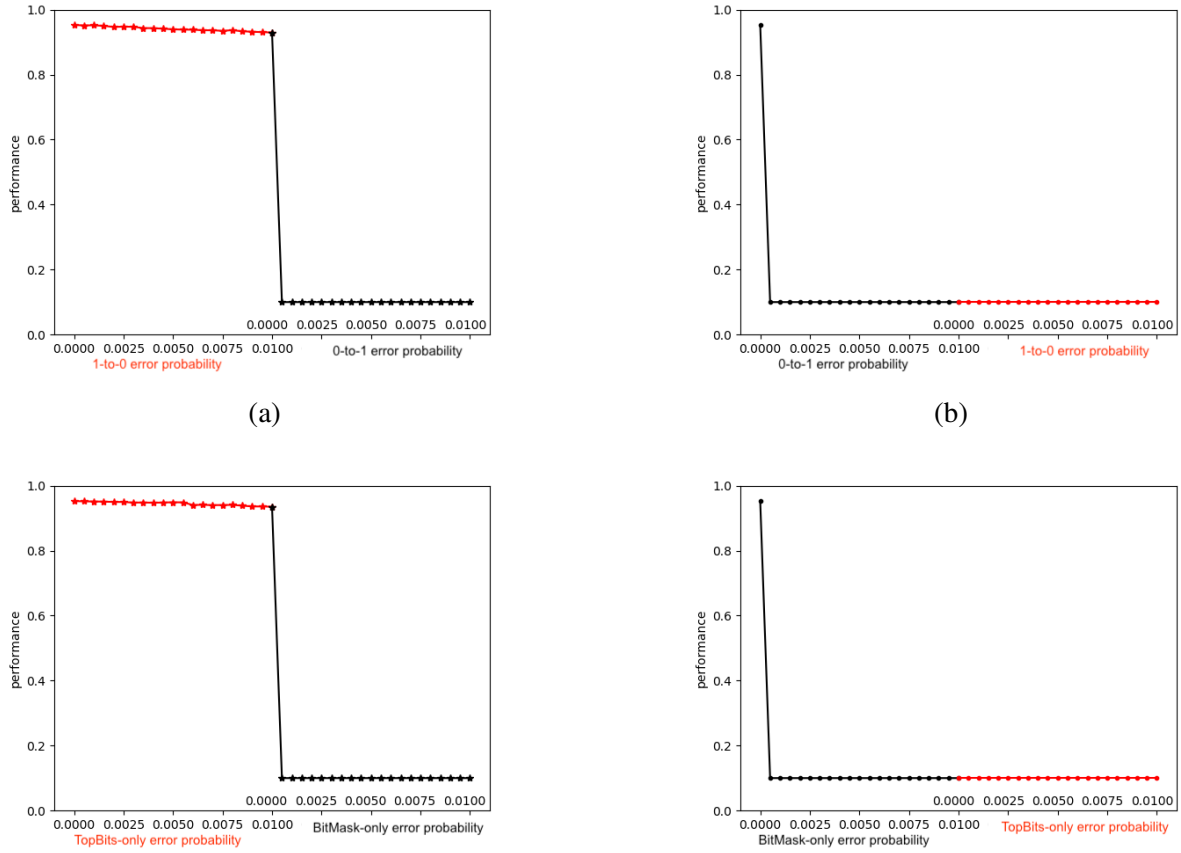


Figure 4.6: How the performance of a neural network changes when errors are added to its bits in two phases. (No bits here are protected by ECC.)

5. CONCLUSIONS

In this work, we use deep learning to selectively protect the weights in neural networks from errors, in order to achieve an optimized redundancy-performance tradeoff. The error-correction scheme is function-oriented: it aims at optimizing the neural network's overall performance, instead of the uncorrectable bit error rates among all the bits after decoding. It studies two important methods for the Selective Protection scheme: the *BitMask* method and the *TopBits* method. Both methods outperform the baseline scheme significantly. And interestingly, it was discovered that sometimes, protecting less significant bits (LSBs) is more important to the neural network's performance than protecting some more significant bits (MSBs).

The proposed error-correction paradigm can be extended in various ways. One interesting extension is to study how errors in different modules in a neural network (including filters, channels, attention modules, *etc*.) affects the neural network's performance, and design error-correction schemes accordingly. They remain as our future research.

Moreover, we can apply different rate of ECCs to the weights represented as bits since the importance of the bits are different. From the observations, the impact of noise added differs layers by layers. We can protect the bits of high importance to the neural network's performance with high rate of ECCs and the bits of low importance with low rate of ECCs. This scheme can further improve the redundancy-performance tradeoff.

The bit-mask vectors that our algorithm chosen are not optimal to the solution. One bottleneck on this algorithm is the deep reinforcement learning algorithm usually overestimates the reward of the observed state and selects a suboptimal bit-mask vectors. To improve the performance of the function approximation, we can develop an algorithm to take a sequence of points and output an estimated function with a set of candidate functions and operations.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[2] J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *CoRR*, vol. abs/1612.00796, 2016.

[3] S. Han, H. Mao, and W. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[4] M. Qin, C. Sun, and D. Vucinic, "Robustness of neural networks against storage media errors," *CoRR*, vol. abs/1709.06173, 2017.

[5] P. Upadhyaya, X. Yu, J. Mink, J. Cordero, P. Parmar, and A. Jiang, "Error correction for noisy neural networks," in *Non-Volatile Memories Workshop*, 2019.

[6] P. Upadhyaya, X. Yu, J. Mink, J. Cordero, P. Parmar, and A. Jiang, "Error correction for hardware-implemented deep neural networks," in *Non-Volatile Memories Workshop*, 2019.

[7] Y. Liu, L. Wei, B. Luo, and Q. Xu, "Fault injection attack on deep neural network," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 131–138, Nov 2017.

[8] A. S. Rakin, Z. He, and D. Fan, "Bit-Flip Attack: Crushing Neural Network with Progressive Bit Search," *arXiv e-prints*, p. arXiv:1903.12269, Mar 2019.

[9] J. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, pp. 5058–5066, 2017.

[10] Y. He and S. Han, "ADC: automated deep compression and acceleration with reinforcement learning," *CoRR*, vol. abs/1802.03494, 2018.

[11] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: hardware-aware automated quantization," *CoRR*, vol. abs/1811.08886, 2018.

[12] A. Gal and M. Szegedy, "Fault tolerant circuits and probabilistically checkable proofs," in *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*, pp. 65–73, June 1995.

[13] V. Choudhary, E. Ledezma, R. Ayyanar, and R. M. Button, "Fault tolerant circuit topology and control method for input-series and output-parallel modular dc-dc converters," *IEEE Transactions on Power Electronics*, vol. 23, pp. 402–411, Jan 2008.

[14] C. E. Stroud, "Reliability of majority voting based vlsi fault-tolerant circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, pp. 516–521, Dec 1994.

[15] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.