Guaranteed Quantization Error Computation for Neural Network Model Compression

Wesley Cooke
School of Computer and Cyber Sciences
Augusta University
Augusta, USA
wcooke@augusta.edu

Zihao Mo
School of Computer and Cyber Sciences
Augusta University
Augusta, USA
zmo@augusta.edu

Weiming Xiang
School of Computer and Cyber Sciences
Augusta University
Augusta, USA
wxiang@augusta.edu

***Abstract*—Neural network model compression techniques can address the computation issue of deep neural networks on embedded devices in industrial systems. The guaranteed output error computation problem for neural network compression with quantization is addressed in this paper. A merged neural network is built from a feedforward neural network and its quantized version to produce the exact output difference between two neural networks. Then, optimization-based methods and reachability analysis methods are applied to the merged neural network to compute the guaranteed quantization error. Finally, a numerical example is proposed to validate the applicability and effectiveness of the proposed approach.**

***Index Terms*—model compression, neural networks, quantization**

I. Introduction

Neural networks have been demonstrated to be powerful and effective tools to solve complex problems such as image processing [1], high-performance adaptive control [2], etc. Due to the increasing complexity of the problems in various applications, the scale and complexity of neural networks also grow exponentially to meet the desired accuracy and performance. Recent progress of machine learning, such as training and using a new generation of large neural networks, heavily depends on the availability of exceptionally large computational resources, e.g., the Transformer model with neural architecture search proposed in [3], if trained from scratch for each case, requires 274,120 hours training on 8 NVIDIA P100 GPUs [4]. Additionally, even for already trained neural networks, the verification process is also quite time- and resource-consuming, e.g., some simple properties in the ACAS Xu neural network of a 5-layer simple structure proposed in [5] need more than 100 hours to be verified. To avoid unaffordable computation when using neural networks, a variety of neural network acceleration and compression methods are proposed such as neural network pruning and quantization, which can significantly reduce the size and memory footprint of neural networks as well as expedite the speed of model inference.

Quantization as a reduction method is mainly concerned with the amount of memory utilized for the learnable parameters of a neural network. The data type for the weights and biases of a typical neural network is usually expressed as 32-bit floating-point values that will carry out millions of floating-point operations during inference time. Quantization aims to shrink the memory footprint of deep neural networks by reducing the number of bits used to store the values for the learnable parameters and activations. This is not only ideal for application scenarios where memory resources may be restricted such as embedded systems or microcontroller environments, but with the selected weight representation you could potentially facilitate faster inference using cheaper arithmetic operations [6]. With the reduction in parameter bit precision, however, it is typical that a quantized neural network will perform worse in terms of accuracy than its non-quantized counterpart using gradient-based learning methods. However, these drops in accuracy are usually considered minimal and worth the given benefit in memory reduction and inference speed-up. There exists much literature describing various techniques of quantization and successful results thereof, including works utilizing stochastic rounding to select weight values beneficial to gradient training [7] and applications on modern deep architectures [8], as well as quantization methods that reduce the number of multiplication operations required during training time [9]. Significant research has also been done on quantization-aware training methods [10], where the loss in accuracy due to bit precision reduction is minimized. Some of these quantization-aware training methods utilize a straight-through gradient estimator (STE) [11] to more appropriately select weights during network training that minimizes accuracy and further reduces computational burden [12].

As quantization methods are used for neural network reduction, there inevitably exist discrepancies between the performances of original and compressed neural networks. In this work, we propose a computationally tractable approach to compute the guaranteed output error caused by quantization. A merged neural network is constructed to generate the output differences between two neural networks, and then reachability analysis on the merged neural network can be performed to obtain the guaranteed error. The remainder of the paper is organized as follows: Preliminaries are given in Section II. The main results on quantization error computation are presented in Section III. A numerical example is given in Section IV. The conclusion is presented in Section V.

This research was supported by the National Science Foundation, under NSF CAREER Award 2143351, and NSF CNS Award no. 2223035.

II. Preliminaries

In this work, we consider a class of fully-connected feedforward neural networks which can be described by the following recursive equations

$$\begin{cases} \mathbf{u}_0 = \mathbf{u} \\ \mathbf{u}_\ell = \phi_\ell(\mathbf{W}_\ell \mathbf{u}_{\ell-1} + \mathbf{b}_\ell),\ \ell = 1, \ldots, L \\ \mathbf{y} = \mathbf{u}_L \end{cases} \tag{1}$$

where $\mathbf{u}_0 = \mathbf{u} \in \mathbb{R}^{n_u}$ is the input vector of the neural network, $\mathbf{y} = \mathbf{u}_L \in \mathbb{R}^{n_y}$ is the output vector of the neural network, $\mathbf{W}_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and $\mathbf{b}_\ell \in \mathbb{R}^{n_\ell}$ are weight matrices and bias vectors for the ℓ-th layer, respectively. $\phi_\ell = [\psi_\ell, \cdots, \psi_\ell]$ is the concatenation of activation functions of the ℓ-th layer in which $\psi_\ell : \mathbb{R} \to \mathbb{R}$ is the activation function, e.g., such as logistic, tanh, ReLU, sigmoid functions. In addition, the input-output mapping of the above neural network $\Phi : \mathbb{R}^{n_u} \to \mathbb{R}^{n_y}$ is denoted in the form of

$$\mathbf{y} = \Phi(\mathbf{u}) \tag{2}$$

where $\mathbf{u} \in \mathbb{R}^{n_u}$ and $\mathbf{y} \in \mathbb{R}^{n_y}$ are the input and output of the neural network, respectively.

A common quantization procedure $\mathsf{Q}(\cdot)$ to map a floating point value r to an integer can be formulated as what follows

$$\mathsf{Q}(r) = \mathsf{int}(r/S) - Z \tag{3}$$

where S is a floating point value as a scaling factor, and Z is an integer value that represents 0 in the quantization policy which could be 0 or other values. The $\mathsf{int} : \mathbb{R} \to \mathbb{Z}$ is the function rounding the floating point value of an integer.

To reduce the size and complexity of the neural network, the quantization procedure is implemented on the neural network parameters, i.e., weights and biases. The quantized version of the neural network (1) is in the form of

$$\begin{cases} \mathbf{u}_0 = \mathbf{u} \\ \mathbf{u}_\ell = \phi_\ell(\mathsf{Q}(\mathbf{W}_\ell)\mathbf{u}_{\ell-1} + \mathsf{Q}(\mathbf{b}_\ell)),\ \ell = 1, \ldots, L \\ \mathbf{y} = \mathbf{u}_L \end{cases} \tag{4}$$

where $\mathsf{Q}(\mathbf{W}_\ell) \in \mathbb{Z}^{n_\ell \times n_{\ell-1}}$ and $\mathsf{Q}(\mathbf{b}_\ell) \in \mathbb{Z}^{n_\ell}$ are the quantized weight matrices and bias vectors for the ℓ-th layer under the quantization process $\mathsf{Q}(\cdot)$. Furthermore, the quantized version of neural network $\Phi : \mathbb{R}^{n_u} \to \mathbb{R}^{n_y}$ is expressed as $\Phi_\mathsf{Q} : \mathbb{Z}^{n_u} \to \mathbb{Z}^{n_y}$ in the form of

$$\mathbf{y} = \Phi_\mathsf{Q}(\mathbf{u}). \tag{5}$$

The quantization can significantly reduce the size and computational complexity of a neural network, e.g., mapping the 32-bit floating point representation to an 8-bit integer representation, leading to smaller models that can fit in hardware with high computational efficiency. However, the price to pay is the loss of performance and precision post-quantization. To formally characterize the performance loss caused by quantization, a reasonable expectation is to compute the quantization error between the neural network and its quantized version.

Definition 1: Given a tuple $\mathbb{M} \triangleq \langle \Phi, \mathsf{Q}, \mathcal{U} \rangle$ where Φ is a neural network defined by (1), Q is the quantization process of (3) producing quantized neural network Φ_Q, and $\mathcal{U} \in \mathbb{R}^{n_u}$ is a compact input set, the guaranteed quantization error is defined by

$$\rho(\mathbb{M}) = \sup_{\mathbf{u} \in \mathcal{U}} \|\Phi(\mathbf{u}) - \Phi_\mathsf{Q}(\mathbf{u})\| \tag{6}$$

where Φ_Q is the quantized neural network of Φ.

Remark 1: The assumption that the input set $\mathcal{U}$ is a compact set is reasonable since neural networks are rarely applied to raw data sets. Instead, standardization and rescaling techniques such as normalization are used which ensure the inputs are always within a compact set such as $[0, 1]$ or $[-1, 1]$. Given the compact input set $\mathcal{U}$ which contains all possible input to the neural network, the guaranteed quantization error $\rho(\mathbb{M})$ characterizes the upper bound for the difference between the outputs of neural network Φ and its quantized version Φ_Q generated from the same inputs in set $\mathcal{U}$, which quantifies the discrepancy caused by the quantization process Q in terms of outputs.

III. Quantization Error Computation

To address the quantization error computation problem, the key is to estimate a $\gamma > 0$ such that $\rho(\mathbb{M}) \leq \gamma$. Due to the complexity of the neural network, it is challenging to estimate the γ directly from the discrepancy of $\Phi(\mathbf{u}) - \Phi_\mathsf{Q}(\mathbf{u})$. Other than directly analyzing the discrepancy of two neural networks, we proposed to construct a new fully-connected neural network $\tilde{\Phi}$ merged from Φ and Φ_Q which is expected to produce the discrepancy of the outputs of two neural networks, i.e., $\tilde{\Phi}(\mathbf{u}) = \Phi(\mathbf{u}) - \Phi_\mathsf{Q}(\mathbf{u})$, and then search for the upper bound of the outputs of the merged neural network.

Given L-layer neural network Φ and quantized Φ_Q, the merged neural network $\tilde{\Phi} : \mathbb{R}^{n_u} \to \mathbb{R}^{n_y}$ is constructed with $L + 1$ layers as follows:

$$\begin{cases} \tilde{\mathbf{u}}_0 = \mathbf{u} \\ \tilde{\mathbf{u}}_\ell = \tilde{\phi}_\ell(\tilde{\mathbf{W}}_\ell \tilde{\mathbf{u}}_{\ell-1} + \tilde{\mathbf{b}}_\ell),\ \ell = 1, \ldots, L+1 \\ \tilde{\mathbf{y}} = \tilde{\mathbf{u}}_{L+1} \end{cases} \tag{7}$$

where

$$\tilde{\mathbf{W}}_\ell = \begin{cases} \begin{bmatrix} \mathbf{W}_1 \\ \mathsf{Q}(\mathbf{W}_1) \end{bmatrix}, & \ell = 1 \\ \begin{bmatrix} \mathbf{W}_\ell & \mathbf{0}_{n_\ell \times n_{\ell-1}} \\ \mathbf{0}_{n_{\ell-1} \times n_\ell} & \mathsf{Q}(\mathbf{W}_\ell) \end{bmatrix}, & 1 < \ell \leq L \\ \begin{bmatrix} \mathbf{I}_{n_y} & -\mathbf{I}_{n_y} \end{bmatrix}, & \ell = L+1 \end{cases} \tag{8}$$

$$\tilde{\mathbf{b}}_\ell = \begin{cases} \begin{bmatrix} \mathbf{b}_\ell \\ \mathsf{Q}(\mathbf{b}_\ell) \end{bmatrix}, & 1 \leq \ell \leq L \\ \begin{bmatrix} \mathbf{0}_{2n_y \times 1} \end{bmatrix}, & \ell = L+1 \end{cases} \tag{9}$$

$$\tilde{\phi}_\ell(\cdot) = \begin{cases} \phi_\ell(\cdot), & 1 \leq \ell \leq L \\ \mathsf{L}(\cdot), & \ell = L+1 \end{cases} \tag{10}$$

where $\mathsf{L}(\cdot)$ is linear transfer function, i.e., $x = \mathsf{L}(x)$.

Theorem 1: Given a tuple $\mathbb{M} \triangleq \langle \Phi, \mathsf{Q}, \mathcal{U} \rangle$ where Φ is a neural network defined by (1), Q is the quantization process of (3), and $\mathcal{U} \in \mathbb{R}^{n_u}$ is a compact input set, the guaranteed quantization error $\rho(\mathbb{M})$ can be computed by

$$\rho(\mathbb{M}) = \sup_{\mathbf{u} \in \mathcal{U}} \left\| \tilde{\Phi}(\mathbf{u}) \right\| \tag{11}$$

where $\tilde{\Phi}$ is a fully-connected neural network defined in (7).

Proof. First, let us consider $\ell = 1$. Given an input $\tilde{\mathbf{u}}_0 = \mathbf{u} \in \mathbb{R}^{n_u}$, one can obtain that

$$\tilde{\mathbf{u}}_1 = \tilde{\phi}_1(\tilde{\mathbf{W}}_1 \tilde{\mathbf{u}}_0 + \tilde{\mathbf{b}}_1) = \begin{bmatrix} \phi_1(\mathbf{W}_1 \tilde{\mathbf{u}}_0 + \mathbf{b}_1) \\ \phi_1(\mathsf{Q}(\mathbf{W}_1)\tilde{\mathbf{u}}_0 + \mathsf{Q}(\mathbf{b}_1)) \end{bmatrix}. \tag{12}$$

Then, we consider $1 < \ell \leq L$. Starting from $\ell = 2$, we have

$$\begin{aligned} \tilde{\mathbf{W}}_2 \tilde{\mathbf{u}}_1 &= \begin{bmatrix} \mathbf{W}_2 & \mathbf{0}_{n_2 \times n_1} \\ \mathbf{0}_{n_2 \times n_1} & \mathsf{Q}(\mathbf{W}_2) \end{bmatrix} \begin{bmatrix} \phi_1(\mathbf{W}_1 \tilde{\mathbf{u}}_0 + \mathbf{b}_1) \\ \phi_1(\mathsf{Q}(\mathbf{W}_1)\tilde{\mathbf{u}}_0 + \mathsf{Q}(\mathbf{b}_1)) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{W}_2 \phi_1(\mathbf{W}_1 \tilde{\mathbf{u}}_0 + \mathbf{b}_1) \\ \mathsf{Q}(\mathbf{W}_2)\phi_1(\mathsf{Q}(\mathbf{W}_1)\tilde{\mathbf{u}}_0 + \mathsf{Q}(\mathbf{b}_1)) \end{bmatrix} \end{aligned}$$

. Furthermore, it leads to

$$\begin{aligned} \tilde{\mathbf{u}}_2 &= \tilde{\phi}_2(\tilde{\mathbf{W}}_2 \tilde{\mathbf{u}}_1 + \tilde{\mathbf{b}}_2) \\ &= \begin{bmatrix} \phi_2(\mathbf{W}_2 \phi_1(\mathbf{W}_1 \tilde{\mathbf{u}}_0 + \mathbf{b}_1) + \mathbf{b}_2) \\ \phi_2(\mathsf{Q}(\mathbf{W}_2)\phi_1(\mathsf{Q}(\mathbf{W}_1)\tilde{\mathbf{u}}_0 + \mathsf{Q}(\mathbf{b}_1)) + \mathbf{b}_2) \end{bmatrix}. \end{aligned}$$

Iterating the above process from $\ell = 2$ to $\ell = L$, the following recursive equation can be derived

$$\tilde{\mathbf{u}}_\ell = \tilde{\phi}_\ell(\tilde{\mathbf{W}}_\ell \tilde{\mathbf{u}}_{\ell-1} + \tilde{\mathbf{b}}_\ell) = \begin{bmatrix} \phi_\ell(\mathbf{W}_\ell \tilde{\mathbf{u}}_{\ell-1} + \mathbf{b}_\ell) \\ \phi_\ell(\mathsf{Q}(\mathbf{W}_\ell)\tilde{\mathbf{u}}_{\ell-1} + \mathsf{Q}(\mathbf{b}_\ell)) \end{bmatrix}$$

where $\ell = 2, \ldots, L$. Together with (12) when $\ell = 1$, it yields that

$$\tilde{\mathbf{u}}_L = \begin{bmatrix} \Phi(\mathbf{u}) \\ \Phi_{\mathsf{Q}}(\mathbf{u}) \end{bmatrix}. \tag{13}$$

Furthermore, when considering the last layer $\ell = L+1$, the following result can be obtained

$$\tilde{\mathbf{u}}_{L+1} = \mathsf{L}\left(\begin{bmatrix} \mathbf{I}_{n_y} & -\mathbf{I}_{n_y} \end{bmatrix} \begin{bmatrix} \Phi(\mathbf{u}) \\ \Phi_{\mathsf{Q}}(\mathbf{u}) \end{bmatrix} \right) = \Phi(\mathbf{u}) - \Phi_{\mathsf{Q}}(\mathbf{u}) \tag{14}$$

where means $\tilde{\Phi}(\mathbf{u}) = \Phi(\mathbf{u}) - \Phi_{\mathsf{Q}}(\mathbf{u})$.

Based on the definition of guaranteed quantization error $\rho(\mathbb{M})$, i.e., Definition 1, we can conclude that

$$\rho(\mathbb{M}) = \sup_{\mathbf{u} \in \mathcal{U}} \|\Phi(\mathbf{u}) - \Phi_{\mathsf{Q}}(\mathbf{u})\| = \sup_{\mathbf{u} \in \mathcal{U}} \left\| \tilde{\Phi}(\mathbf{u}) \right\|. \tag{15}$$

The proof is complete. □

Remark 2: Theorem 1 implies that we can analyze the merged neural network $\tilde{\Phi}$ to compute the quantization error between neural network Φ and its quantized version Φ_{Q}. This result facilitates the computation process by employing those analyzing tools, such as optimization and reachability analysis tools, for merged neural network $\tilde{\Phi}$.

- Using the interval arithmetic for neural network, we can employ Moore-Skelboe Algorithm [13] to search upper bound of $||\tilde{\Phi}(\mathbf{u})||$ subject to $\mathbf{u} \in \mathcal{U}$ where $\mathcal{U}$ is a compact set. The key to implement Moore-Skelboe Algorithm is to construct the interval extension of $[\tilde{\Phi}] : \mathbb{IR}^{n_u} \to \mathbb{IR}^{n_y}$. First, from Theorem 1 in [14] under the assumption that activation functions are monotonically increasing, the interval extension of merged neural network $[\tilde{\Phi}]$ can be constructed as

$$[\tilde{\Phi}] = [\tilde{\Phi}^-, \tilde{\Phi}^+] \tag{16}$$

where $\tilde{\Phi}^-$ and $\tilde{\Phi}^+$ are left (limit inferior) and right (limit superior) bounds of interval $[\tilde{\Phi}]$ that are defined as follows

$$\tilde{\Phi}^- : \begin{cases} \tilde{\mathbf{u}}_0^- = \mathbf{u}^- \\ \tilde{\mathbf{u}}_\ell^- = \tilde{\phi}_\ell \left(\begin{bmatrix} \tilde{\mathbf{W}}_\ell^- & \tilde{\mathbf{W}}_\ell^+ \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{u}}_{\ell-1}^+ \\ \tilde{\mathbf{u}}_{\ell-1}^- \end{bmatrix} + \tilde{\mathbf{b}}_\ell \right) \\ \tilde{\mathbf{y}}^- = \tilde{\mathbf{u}}_{L+1}^- \end{cases}$$

$$\tilde{\Phi}^+ : \begin{cases} \tilde{\mathbf{u}}_0^+ = \mathbf{u}^+ \\ \mathbf{u}_\ell^+ = \tilde{\phi}_\ell \left(\begin{bmatrix} \tilde{\mathbf{W}}_\ell^- & \tilde{\mathbf{W}}_\ell^+ \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{u}}_{\ell-1}^- \\ \tilde{\mathbf{u}}_{\ell-1}^+ \end{bmatrix} + \tilde{\mathbf{b}}_\ell \right) \\ \tilde{\mathbf{y}}^+ = \tilde{\mathbf{u}}_{L+1}^+ \end{cases}$$

in which $\mathcal{U} \subseteq [\mathbf{u}] = [\mathbf{u}^-, \mathbf{u}^+]$, and

$$\mathbf{W}_\ell^- = [\underline{w}_\ell^{i,j}],\ \underline{w}_\ell^{i,j} = \begin{cases} w_\ell^{i,j} & w_\ell^{i,j} < 0 \\ 0 & w_\ell^{i,j} \geq 0 \end{cases} \tag{17}$$

$$\mathbf{W}_\ell^+ = [\overline{w}_\ell^{i,j}],\ \overline{w}_\ell^{i,j} = \begin{cases} w_\ell^{i,j} & w_\ell^{i,j} \geq 0 \\ 0 & w_\ell^{i,j} < 0 \end{cases} \tag{18}$$

with $w_\ell^{i,j}$, $\underline{w}_\ell^{i,j}$, and $\overline{w}_\ell^{i,j}$ being the elements in i-th row and j-th column of matrix $\mathbf{W}_\ell$, $\mathbf{W}_\ell^-$, and $\mathbf{W}_\ell^+$. With the above tractable calculation of $\tilde{\Phi}^-$ and $\tilde{\Phi}^+$, we can perform Moore-Skelboe Algorithm to compute guaranteed quantization error $\rho(\mathbb{M})$.

- Under the framework of reachability analysis of neural networks, the guaranteed quantization error computation problem can be turned into a reachable set computation problem for merged neural network $\tilde{\Phi}$. Given the input set $\mathcal{U}$, the following set

$$\mathcal{Y} = \left\{ \tilde{\mathbf{y}} \in \mathbb{R}^{n_y} \mid \tilde{\mathbf{y}} = \tilde{\Phi}(\mathbf{u}),\ \mathbf{u} \in \mathcal{U} \right\} \tag{19}$$

is called the output set of neural network (1). The guaranteed quantization error $\rho(\mathbb{M})$ can be obtained by

$$\rho(\mathbb{M}) = \max\{\mathbf{y} \mid \tilde{\mathbf{y}} \in \mathcal{Y}\}. \tag{20}$$

The key step is the computation for the reachable set $\mathcal{Y}$. This can be efficiently done through neural network reachability analysis. There exist a number of verification tools for neural networks available for the reachable set computation. The neural network reachability analysis tool can produce the reachable set $\mathcal{Y}$ in the form of a union of polyhedral sets such as NNV [15], veritex [16], etc. The IGNNV tool computes the reachable set $\mathcal{Y}$ as a union of interval sets [14], [17]. With the reachable set $\mathcal{Y}$, the guaranteed quantization error $\rho(\mathbb{M})$ can be easily obtained by searching for the maximal value of $\|\tilde{\mathbf{y}}\|$ in

TABLE I
NEURAL NETWORK MODEL MEMORY SIZES

Neural Networks	Size (KB)
Original Neural Network ($1 \times 50 \times 50 \times 50 \times 1$)	35.0
Quantized Neural Network ($1 \times 50 \times 50 \times 50 \times 1$)	19.6

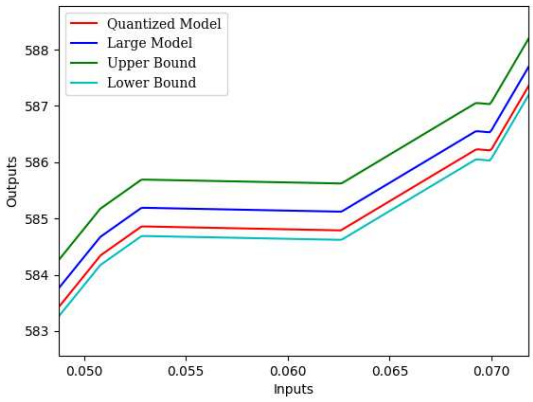

Fig. 1. The lower- and upper bounds of outputs of quantized neural networks based on a guaranteed quantization error.

$\mathcal{Y}$, e.g., testing throughout a finite number of vertices in the interval or polyhedral sets.

IV. NUMERICAL EXAMPLE

To verify the effectiveness of the quantization error computation, a numerical example is used. First, a large neural network, Φ, is generated such that it has a 1-D input layer, three hidden layers with 50 neurons in each layer, and a 1-D output layer. Each layer has an activation function of ReLU except for the output layer with a linear function. The weights and biases were randomly initialized, but with the condition that they were normally distributed with a mean of zero and a standard deviation of one.

After generating Φ, a quantization method was applied to reduce the size of the weights and biases, and to introduce a slight reduction in accuracy. This quantized network is called Φ_{Q}. While there exist several quantization methods and tools to quantize networks, a basic technique that truncates the weights and biases to 4 decimal places is used in this numerical example.

Next, a merged network $\tilde{\Phi}$ was constructed from Φ and Φ_{Q} according to (7). The veritex neural network reachability tool [16] computed the reachable output set of $\tilde{\Phi}$ given the input interval normalized as $[0, 1]$. Finally, the quantization error was obtained using (20) which is $\rho(\mathbb{M}) = 0.5008$. Using this error, the lower and upper bound can be constructed using the following formula: $\Phi(u) \pm \rho(\mathbb{M})$ as shown in Fig. 1. Please note that this quantization error is very small compared with the range of outputs. Thus, to provide more detail, the figure has been zoomed into an appropriate scale. Moreover, the memory sizes of models are shown in Table I.

V. CONCLUSIONS

This paper addressed the guaranteed output error computation problem for neural network compression with quantization. Based on the original neural network and its compressed version resulted from quantization, a merged neural network computation framework is developed, which can utilize optimization-based methods and reachability analysis methods to compute the guaranteed quantization error. At last, numerical examples are proposed to validate the applicability and effectiveness of the proposed approach. Future work will be expanded to include more complex and various neural network architectures such as convolutional neural networks.

REFERENCES

[1] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[2] K. Hunt, D. Sbarbaro, R. Żbikowski, and P. Gawthrop, "Neural networks for control systems—a survey," *Automatica*, vol. 28, no. 6, pp. 1083–1112, 1992.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[4] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 3645–3650.

[5] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 97–117.

[6] W. Roth, G. Schindler, M. Zöhrer, L. Pfeifenberger, R. Peharz, S. Tschiatschek, H. Fröning, F. Pernkopf, and Z. Ghahramani, "Resource-efficient neural networks for embedded systems," *arXiv preprint arXiv:2001.03048*, 2020.

[7] M. Höhfeld and S. E. Fahlman, "Probabilistic rounding in neural network learning with limited precision," *Neurocomputing*, vol. 4, no. 6, pp. 291–299, 1992.

[8] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, 2015, pp. 1737–1746.

[9] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," *arXiv preprint arXiv:1510.03009*, 2015.

[10] E. Park, S. Yoo, and P. Vajda, "Value-aware quantization for training and inference of neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[11] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[12] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[13] R. E. Moore and H. Ratschek, "Inclusion functions and global optimization ii," *Mathematical Programming*, vol. 41, no. 1, pp. 341–356, 1988.

[14] W. Xiang, H.-D. Tran, X. Yang, and T. T. Johnson, "Reachable set estimation for neural network control systems: A simulation-guided approach," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 5, pp. 1821–1830, 2021.

[15] H.-D. Tran, X. Yang, D. M. Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, "NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems," *arXiv preprint arXiv:2004.05519*, 2020.

[16] X. Yang, T. Yamaguchi, B. Hoxha, D. Prokhorov, and T. T. Johnson, "Veritex: Tool for reachability analysis and repair of neural networks." [Online]. Available: https://github.com/verivital/veritex

[17] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 11, pp. 5777–5783, 2018.