Weight-Rounding Error in Deep Neural Networks

Jiří Šíma[0000−0001−8248−9425] (✉) and Petra Vidnerová[0000−0003−3879−3459]

Institute of Computer Science of the Czech Academy of Sciences, Prague 8, Czechia
{sima,petra}@cs.cas.cz

Abstract. Current AI technologies based on deep neural networks (DNNs) are computationally extremely demanding, which limits their widespread deployment in embedded devices with constrained energy resources (e.g. battery-powered smartphones). One possible approach to solving this problem is to reduce the precision of weight parameters, which can save an enormous amount of energy for computation and data transfer at the cost of only a small loss in inference accuracy. In this paper, we provide a theoretical analysis of the effect of any weight rounding (e.g. reduced bitwidth) in a trained DNN on its output. We first derive a global upper bound on the output error of DNN (under the L_1 norm) caused by the weight rounding for all inputs from a bounded domain in the worst case, which turns out to be overestimated for practical use. We prove that computing this maximum error is NP-hard for a given weight rounding even for two layers, which follows from the NP-hardness of neuron state domains. Based on the concept of so-called shortcut weights, we propose a method called AppMax that estimates this error using linear programming on convex polytopes around test/training data points, which works for any approximation of DNN (e.g. including pruning). The AppMax method was extensively tested on fully connected and convolutional neural networks (trained on the MNIST database) for decreasing bitwidth of weights. The experiments demonstrate a clear improvement in the error guarantees provided by this method, which can be used to evaluate different approximation strategies and identify those that best balance accuracy and energy efficiency.

Keywords: Deep neural networks · Weight rounding · Maximum error bounds · NP-hardness · Linear programming.

1 Introduction

Deep neural networks (DNN) stands at the forefront of both research and practical applications in artificial intelligence (AI), including large language models, image recognition, computer vision, speech recognition, robotics, etc. An increasing number of embedded devices rely on DNNs to deliver sophisticated services, such as autonomous surveillance systems utilizing advanced object recognition, personal assistants employing machine translation, smart healthcare applications. Nevertheless, remarkable performance on these tasks comes with increased computational demands, posing significant challenges for deploying DNNs on

resource-constrained edge devices. The escalated computational requirements of DNNs which typically consist of tens of layers, hundreds of thousands of neurons, and tens of millions of weight parameters, naturally lead to increased battery consumption, which is a major bottleneck to the development of smart wearable electronics, such as smartphones, smart glasses, or voice assistants. In light of these considerations, addressing the energy consumption of DNN implementations emerges as a topic of paramount importance.

Recent research has focused on developing methods that enable energy-efficient processing of DNNs [26]. There are basically two main approaches to reduce energy costs of DNNs. First, computational requirements are addressed by *hardware design* utilizing specialized accelerators tailored for DNN inference [11, 23] that employ massive parallelism. DNNs are implemented on various hardware platforms including GPUs [32], FPGAs [18], in-memory computing architectures [19], etc. which share asymptotic energy complexity bounds [25, 24].

The second approach is suitable for error-tolerant applications such as image classification where enormous amount of energy can be saved at the cost of only a small loss in accuracy by using *approximate computing* methods [2, 6, 15–17, 27]. One possibility is to reduce the number of operations and model size which is based on techniques such as compression [4], pruning [31], and compact network architectures [12]. Another possibility is to reduce the precision of the used arithmetics which includes conversion from floating point to fixed point [30], reducing the bitwidth [20], nonuniform quantization [14], weight sharing [8], and approximate multiplication circuits [1]. For example, an 8-bit fixed-point multiply consumes 15.5 or 18.5 times less energy than a 32-bit fixed-point or floating-point multiply, respectively [9].

It has been empirically observed that the energy consumption for DNN inference is predominantly due to both numerical computation and data moves in memory where the later can achieve 70% of the total energy cost [31]. Both these energy components can be significantly decreased by reducing the precision of DNN weight parameters. In this paper, we theoretically analyze the effect of weight rounding in a trained DNN on its output. This post-training rounding is specified by individual weight deviations and can thus be generated by any method, such as those referenced above (e.g., reduced bitwidth, quantization). For the purposes of error analysis, we formalize a model of feedforward (acyclic) DNNs composed of ReLU gates employing the rectified linear unit activation function (Section 2), which appears also to cover a widespread important DNN class of convolutional neural networks (CNNs).

First, we derive a global worst-case upper bound on the DNN output error that is caused by a given weight rounding, which is valid for all inputs from a bounded domain (Section 3). The error of DNN outputs is measured by the L_1 norm which makes the analysis applicable to regression tasks. The main idea is that the output error of any individual neuron is bounded for *all* its inputs taken from a previously estimated interval state domains, which is propagated feedforwardly through the network. Nevertheless, our experiments show that this worst-case upper bound on the DNN output error turns out to be overestimated

for practical tasks. It could be improved by refining the estimation of neuron state domains, which, however, are shown to be NP-hard to compute even for two-layer networks (Section 4). Since the maximum of the DNN output can be attained at any interior point of the input domain, we prove this result for both binary and real inputs. As a consequence, we obtain NP-hardness also for computing the global maximum error for a given weight rounding even for two layers.

Furthermore, we introduce a concept of so-called shortcut weights which are coefficients of the linear dependence of DNN outputs on its inputs for fixed saturations of neuron states, due to the ReLU activation function is piecewise linear. Based on these shortcut weights, we propose a method called AppMax that estimates the maximum error for a given weight rounding using linear programming on convex polytopes surrounding test/training data points (Section 5). The method is also adapted to classification tasks and it is applicable not only to weight rounding but it provides the maximum error for any approximation of a DNN by another network, e.g., created by pruning. We have tested the AppMax method on fully connected and convolutional neural networks trained on the MNIST database for decreasing bitwidth of weights (Section 6). The presented experiments demonstrate that the AppMax method provides more confident estimates of the maximum error than those on the test data points only, which is already achieved with fewer data.

A related recent study [3] develops a theoretical framework for analyzing the numerical stability of DNNs with differentiable activation functions (e.g., hyperbolic tangent) under floating-point arithmetic, using backward error analysis and condition numbers. While its focus is on global perturbation models, our work complements this by introducing the practically applicable AppMax method for estimating approximation errors of piecewise linear DNNs under arbitrary post-training modifications, including but not limited to weight rounding. The proposed AppMax method enables systematic comparison of approximation strategies and helps identify those that offer the best trade-off between accuracy and performance. Further research in this direction could facilitate the development of techniques for identifying DNN components suitable for approximation or removal, aiming to reduce energy consumption while maintaining explicit and reliable error guarantees.

2 A Formal Model of NNs and Weight Rounding

For the error analysis of weight rounding, we define a formal model of (artificial) feedforward neural networks (NNs) with the ReLU (rectified linear unit) activation function, which covers commonly used DNNs such as convolutional neural networks (CNNs). The architecture of a NN $\mathcal{N}$ is a connected directed acyclic graph (V, E) where $E \subset V \times V$, which is composed of units, called neurons, whose real states (outputs) are denoted as y_j for $j \in V$. This includes a set of n input neurons $X = \{1, \ldots, n\} \subseteq V$ that serve only for presenting an external real input $(x_1, \ldots, x_n) \in \mathbb{R}^n$ to $\mathcal{N}$, that is $y_j = x_j$ for $j \in X$, whereas a set of m

output neurons, $Y \subseteq V' = V \setminus X$ provides the output $\mathcal{N}(x_1, \ldots, x_n) \in \mathbb{R}^m$ from $\mathcal{N}$ for this input. For any neuron $j \in V$, we denote by $j_{\leftarrow} = \{i \in V \mid (i,j) \in E\}$ the set of units in $\mathcal{N}$ from which connections (edges) lead to j, which represent the inputs to j. Thus, we assume $j_{\leftarrow} = \emptyset$ for $j \in X$, and $j_{\leftarrow} \cap Y = \emptyset$ for $j \in V$.

For any non-input neuron $j \in V'$ and its input $i \in j_{\leftarrow}$, let $w_{ji} \in \mathbb{R}$ be a real weight associated with the connection $(i,j) \in E$, whereas formally $w_{ji} = 0$ for $i \in V \setminus j_{\leftarrow}$. In addition, $w_{j0} \in \mathbb{R}$ denotes its real bias, which, as usual, can be viewed as a weight of an edge $(0,j) \in E$ leading from an additional formal input neuron $0 \in X$ to j, whose state $y_0 = 1$ is constantly one and $0 \in j_{\leftarrow}$ for every $j \in V'$ such that $w_{j0} \neq 0$. Then the excitation ξ_j of neuron $j \in V'$ is evaluated as a weighted sum of its inputs:

$$\xi_j = \sum_{i \in j_{\leftarrow}} w_{ji} y_i \,, \tag{1}$$

provided that the states y_i have already been computed for all units $i \in j_{\leftarrow}$, after an external input $(x_1, \ldots, x_n)$ to $\mathcal{N}$ was given at the beginning. Then the output y_j from neuron $j \in V'$ is computed by applying the ReLU activation function $R : \mathbb{R} \to \mathbb{R}$ to its excitation ξ_j,

$$y_j = R(\xi_j) = \max(0, \xi_j) \,. \tag{2}$$

Alternatively, for *classification* tasks, the states y_j of at least two output neurons $j \in Y$ normalize their excitations into a categorical probability distribution by using the softmax function

$$y_j = \frac{e^{\xi_j}}{\sum_{k \in Y} e^{\xi_k}} \in (0,1) \,. \tag{3}$$

while the identity $y_j = \xi_j$ for $j \in Y$ can be employed for *regression* (to allow negative outputs).

The convolutional layers in CNNs are at times interlaced by max pooling layers whose units j implement the maximum of its inputs, $y_j = \max_{i \in j_{\leftarrow}} y_i$. Note that $y_i \geq 0$ for $i \in V'$ due to (2), and we will assume $y_i \geq 0$ for $i \in X$ without loss of generality (see below). Then such a max pooling unit can be replaced in $\mathcal{N}$ by a subnetwork composed of neurons that compute their states according to (1) and (2), since the maximum of two numbers $\max(x,y) = R(x-y) + y$ for $x, y \geq 0$, can be used for evaluating the maximum of $|j_{\leftarrow}|$ nonnegative inputs (e.g., the maxima of pairs is used to compute the maxima of fours, eights, sixteens, etc.). The alternative average pooling unit j that computes the average of its nonnegative inputs, $y_j = \frac{1}{|j_{\leftarrow}|} \sum_{i \in j_{\leftarrow}} y_i$, can be viewed as a neuron $j \in V'$ with the bias $w_{j0} = 0$ and weights $w_{ji} = 1/|j_{\leftarrow}|$ for $i \in j_{\leftarrow} \setminus \{0\}$. Thus, we will hereafter assume without loss of generality that $\mathcal{N}$ does not contain pooling layers.

Suppose that the weights (including the biases) in $\mathcal{N}$ are rounded, e.g. to a given number of binary digits in their floating-point representations, which can be expressed as

$$\widetilde{w_{ji}} = w_{ji} + \delta_{ji} \quad \text{for } j \in V' \;\&\; i \in j_{\leftarrow} \tag{4}$$

where $\delta_{ji} \in \mathbb{R}$ is a real rounding error of weight w_{ji}. Denote by $\widetilde{\xi}_j$ and $\widetilde{y}_j$ the excitation and output of $j \in V'$, respectively, that are computed by using the rounded weights (4). We will be interested in the effect of this weight rounding on the output of $\mathcal{N}$, which is measured for regression tasks by the L_1 norm as the sum of excitation deviations of output neurons for given external inputs:

$$E(x_1, \ldots, x_n) = \sum_{j \in Y} \left| \xi_j - \widetilde{\xi}_j \right| . \tag{5}$$

3 The Worst-Case Error Bounds for Weight Rounding

For an unbounded external input $(x_1, \ldots, x_n) \in \mathbb{R}^n$ to $\mathcal{N}$, the output error (5) caused by weight rounding (4) can be arbitrarily large. In practical applications, however, this input is usually taken from some bounded interval domain:

$$a_i \leq x_i \leq b_i \quad \text{for } i \in X \tag{6}$$

(particularly, $a_0 = x_0 = b_0 = 1$ for the formal input $0 \in X$ corresponding to biases w_{j0} for $j \in V'$) which ensures $a_i \leq y_i \leq b_i$ for $i \in X$. In addition, we assume without loss of generality that $a_i = 0$ and $b_i = 1$ for each input neuron $i \in X \setminus \{0\}$, since external inputs can be linearly mapped onto $[0, 1]^n$ by appropriately adjusting the corresponding weights.

Then we can estimate the state intervals $[a_j, b_j]$ also for non-input neurons $j \in V'$ by propagating the input domain (6) through $\mathcal{N}$ so that

$$a_j \leq y_j \leq b_j \quad \text{for } j \in V' , \tag{7}$$

provided that $a_i \leq y_i \leq b_i$ for all its inputs $i \in j_{\leftarrow}$. This can be ensured by

$$a_j = R(a'_j) , \quad b_j = R(b'_j) \quad \text{for } j \in V' , \quad \text{where} \tag{8}$$

$$a'_j = \sum_{\substack{i \in j_{\leftarrow} \\ w_{ji} < 0}} w_{ji} b_i + \sum_{\substack{i \in j_{\leftarrow} \\ w_{ji} > 0}} w_{ji} a_i , \quad b'_j = \sum_{\substack{i \in j_{\leftarrow} \\ w_{ji} < 0}} w_{ji} a_i + \sum_{\substack{i \in j_{\leftarrow} \\ w_{ji} > 0}} w_{ji} b_i \quad \text{for } j \in V' . \tag{9}$$

Furthermore, we assume without loss of generality that $a_j = 0$ and $b_j > 0$ for $j \in V \setminus (Y \cup \{0\})$, since any non-input neuron $j \in V'$ with $a_j > 0$ or $b_j = 0$ can be eliminated from $\mathcal{N}$ as its output is linear, $y_j = \xi_j$, or zero, $y_j = 0$, respectively, due to (7) and (2). Note that the interval bounds (9) represent the possible worst case only for one neuron while they are overestimated when combined for more consecutive neurons in deep $\mathcal{N}$ where their states usually cannot reach the interval endpoints (8) for any external input satisfying (6). We will show in Section 4 that it is actually NP-hard to find the tight bounds in (7) already for two-layer $\mathcal{N}$.

For weight rounding (4), the states $\widetilde{y}_i$ of inputs $i \in j_{\leftarrow}$ to a non-input neuron $j \in V'$ are supposed to be within intervals around their precise values y_i as

$$y_i + \alpha_i \leq R(y_i + \alpha_i) \leq \widetilde{y}_i \leq y_i + \beta_i \quad \text{for } i \in j_{\leftarrow} , \tag{10}$$

where $\alpha_i \leq 0 \leq \beta_i$. Clearly, the states $y_i = \widetilde{y}_i$ of input units $i \in X$ are not affected by weight rounding, which means $\alpha_i = \beta_i = 0$ for $i \in X$. The main idea of estimating the impact of weight rounding on the non-input neuron $j \in V'$ in the worst case, is to bound $\widetilde{y}_j$ so that for some $\alpha_j \leq 0 \leq \beta_j$,

$$y_j + \alpha_j \leq R(y_j + \alpha_j) \leq \widetilde{y}_j \leq y_j + \beta_j \tag{11}$$

for every $\widetilde{y}_i \in [R(y_i + \alpha_i), y_i + \beta_i]$ in (10), over all $y_i \in [a_i, b_i]$ (where $a_i = 0$ and $b_i > 0$) given by (6) for $i \in j_{\leftarrow} \cap X$ or by (8) for $i \in j_{\leftarrow} \cap V'$. This can be achieved by

$$\alpha_j = \min(0, \alpha'_j) \leq 0\,, \quad \beta_j = \max(0, \beta'_j) \geq 0 \tag{12}$$

where $\xi_j + \alpha'_j \leq \widetilde{\xi}_j \leq \xi_j + \beta'_j$ is ensured by

$$\alpha'_j = \delta_{j0} + \sum_{\substack{i \in j_{\leftarrow} \\ \delta_{ji} < 0}} \delta_{ji} b_i + \sum_{\substack{i \in j_{\leftarrow} \\ \widetilde{w_{ji}} > 0}} \widetilde{w_{ji}} \alpha_i + \sum_{\substack{i \in j_{\leftarrow} \\ \widetilde{w_{ji}} < 0}} \widetilde{w_{ji}} \beta_i \tag{13}$$

$$\beta'_j = \delta_{j0} + \sum_{\substack{i \in j_{\leftarrow} \\ \delta_{ji} > 0}} \delta_{ji} b_i + \sum_{\substack{i \in j_{\leftarrow} \\ \widetilde{w_{ji}} < 0}} \widetilde{w_{ji}} \alpha_i + \sum_{\substack{i \in j_{\leftarrow} \\ \widetilde{w_{ji}} > 0}} \widetilde{w_{ji}} \beta_i \tag{14}$$

for $j \in V'$.

We show that (12)–(14) meets (11). It follows from (10) and (4) that

$$\begin{aligned} \widetilde{\xi}_j &= \sum_{i \in j_{\leftarrow}} \widetilde{w_{ji}}\, \widetilde{y}_i \geq \sum_{\substack{i \in j_{\leftarrow} \\ \widetilde{w_{ji}} \geq 0}} \widetilde{w_{ji}}(y_i + \alpha_i) + \sum_{\substack{i \in j_{\leftarrow} \\ \widetilde{w_{ji}} < 0}} \widetilde{w_{ji}}(y_i + \beta_i) \\ &= \sum_{i \in j_{\leftarrow}} (w_{ji} + \delta_{ji}) y_i + \sum_{\substack{i \in j_{\leftarrow} \\ \widetilde{w_{ji}} > 0}} \widetilde{w_{ji}} \alpha_i + \sum_{\substack{i \in j_{\leftarrow} \\ \widetilde{w_{ji}} < 0}} \widetilde{w_{ji}} \beta_i \geq \xi_j + \alpha'_j \end{aligned} \tag{15}$$

according to (6), (7), $a_i = 0$ for $i \in j_{\leftarrow} \setminus \{0\}$, and (13), which gives

$$\widetilde{y}_j = R(\widetilde{\xi}_j) \geq R(\xi_j + \alpha'_j) \geq R(\xi_j + \alpha_j) \geq R(\xi_j) + \alpha_j = y_j + \alpha_j \tag{16}$$

due to (12). Similarly, $\widetilde{\xi}_j \leq \xi_j + \beta'_j$ which implies $\widetilde{y}_j \leq y_j + \beta_j$. This completes the proof of (11).

The theoretical estimates (12)–(14) including (8) and (9), can be used to compute an upper bound on the weight-rounding error (5) of $\mathcal{N}$ in the worst case, which is valid for any external input from the interval domain (6):

$$\max_{(x_1,\ldots,x_n) \in [0,1]^n} E(x_1, \ldots, x_n) \leq \sum_{j \in Y} \max(-\alpha'_j, \beta'_j)\,. \tag{17}$$

Nevertheless, practical experiments show that these bounds are very conservative. For example, we tested a NN (see Net1 in Section 6) with only three fully connected layers whose weights were rounded to be represented by 16 bits. The experiment has shown that the intervals $[\alpha_j, \beta_j]$ for $j \in V$, tend to get larger by magnitude every next layer (see Tab. 1). For comparison, the actual values of error (5) are shown in Section 6 to be less than 0.1 for all test data points. Therefore, this approach turns out to be infeasible for practical usage, since the interval bounds are highly overestimated.

Table 1. The smallest and widest intervals $[\alpha_j, \beta_j]$ according to (12)–(14) for each of the three layers of the fully-connected NN Net1.

Layer	Smallest $[\alpha_j, \beta_j]$	Widest $[\alpha_j, \beta_j]$
1	[-0.0016, 0.0028]	[-0.0142, 0.0157]
2	[-2.0662, 2.0615]	[-2.6336, 2.6642]
3	[-57.5910, 58.6081]	[-84.9428, 85.1832]

4 Computing the Maximum Error is NP-hard

In Section 3, the worst-case error bound (17) computed by (12)–(14) is overestimated also because it is based on the very conservative estimates of the interval boundaries of neuron states (8), (9). Thus, there is a natural issue whether these estimates could be improved. This can be formulated as the problem of finding the maximum output for a given NN. Namely, we are given $\mathcal{N}$ with one output ($|Y| = 1$) and $|X| = n$ input neurons, and a positive real constant $M > 0$. The *maximum output* problem (MO) is to decide whether there is an external input $(x_1, \ldots, x_n) \in [0, 1]^n$ to $\mathcal{N}$ such that $\mathcal{N}(x_1, \ldots, x_n) \geq M$. However, this problem is proven to be computationally hard by reduction from the *Boolean satisfiability* problem SAT (proof omitted):

Theorem 1. *MO is NP-hard for two-layer NNs with binary inputs from* $\{0, 1\}^n$.

Nevertheless, the maximum output value of NNs can also be attained at any interior point of the external input domain $[0, 1]^n$. Consider a simple example of a NN $\mathcal{N}_\mu$ with a real parameter $\mu \in (0, 1)$, that is composed of two layers with one input neuron ($n = 1$) for external inputs $x \in [0, 1]$, one unit in the first layer, and one output neuron which computes the function $\mathcal{N}_\mu(x) = R\big(x - R(\frac{x}{\mu} - 1)\big)$. Observe that $\mathcal{N}_\mu(x) = x$ for every x such that $0 \leq x \leq \mu$ due to $R(\frac{x}{\mu} - 1) = 0$ in this case. On the other hand, for every x satisfying $\mu < x \leq 1$ we have $R(\frac{x}{\mu} - 1) = \frac{x}{\mu} - 1$ which, for $\mathcal{N}_\mu(x) > 0$, implies $\mathcal{N}_\mu(x) = R\big(x - (\frac{x}{\mu} - 1)\big) = \frac{\mu - 1}{\mu}\, x + 1 < \mu$ due to $x > \mu$ and $0 < \mu < 1$. It follows that the maximum output value $\max_{x \in [0,1]} \mathcal{N}_\mu(x) = \mu$ is attained at the interior point $x = \mu \in (0, 1)$. Therefore, we generalize Theorem 1 to the *real* external input domain.

Theorem 2. *MO is NP-hard for two-layer NNs with real inputs from* $[0, 1]^n$ *(remains NP-hard even if the excitation of the output neuron is nonnegative).*

Proof. We will use a reduction from the known NP-complete *Exactly-1 Positive 3-SAT* problem (1-in-3-SAT+) [22] which is a variant of SAT restricted to Boolean formulas in conjunctive normal form with 3 positive literals (i.e. variables) per clause, to decide whether there exists an assignment to the variables so that each clause has 1 assigned to exactly one variable (and thus 0 assigned to exactly two variables). For any such 1-in-3-SAT+ instance φ over n variables $y_1, \ldots, y_n \in \{0, 1\}$ with r clauses $C_1, \ldots, C_r \subseteq \{y_1, \ldots, y_n\}$ where $|C_j| = 3$ for all $j \in \{1, \ldots, r\}$, we will construct a NN $\mathcal{N}_\varphi^M$ so that there exists an assignment to variables $y_i = x_i \in \{0, 1\}$ for every $i \in \{1, \ldots, n\}$, satisfying φ such that

$|\{y_i \in C_j \mid x_i = 1\}| = 1$ for all $j \in \{1, \ldots, r\}$ iff the output of $\mathcal{N}_\varphi^M$ reaches a given real bound $M > 0$ for some external real input.

The NN $\mathcal{N}_\varphi^M$ is composed of two layers with n input neurons for external real inputs $(x_1, \ldots, x_n) \in [0,1]^n$, representing the n Boolean variables $y_1, \ldots, y_n \in \{0,1\}$ of φ, $3r$ units in the first layer corresponding to the 3 variables per each of the r clauses $C_1, \ldots, C_r$ in φ, and one output neuron on the top which computes the function $\mathcal{N}_\varphi^M(x_1, \ldots, x_n) = R\big(\sum_{j=1}^r \frac{M}{r} Y_j\big)$ where

$$Y_j = R(x_{j1} - x_{j2} - x_{j3}) + R(x_{j2} - x_{j1} - x_{j3}) + R(x_{j3} - x_{j1} - x_{j2}) \geq 0 \quad (18)$$

for $C_j = \{y_{j1}, y_{j2}, y_{j3}\} \subseteq \{y_1, \ldots, y_n\}$, for every $j \in \{1, \ldots, r\}$.

For each $j \in \{1, \ldots, r\}$, suppose at least two of the 3 terms $x_{j1} - x_{j2} - x_{j3}$, $x_{j2} - x_{j1} - x_{j3}$, and $x_{j3} - x_{j1} - x_{j2}$ in (18) are nonnegative, say $x_{j1} - x_{j2} - x_{j3} \geq 0$ and $x_{j2} - x_{j1} - x_{j3} \geq 0$ (similarly for the other pairs of terms), which sums to $-2x_{j3} \geq 0$ implying $x_{j3} = 0$ and hence, $x_{j1} - x_{j2} = x_{j2} - x_{j1} = 0$. Thus, we have $Y_j = R(-x_{j1} - x_{j2}) = 0$ according to (18). On the other hand, if all these 3 terms are negative, then clearly $Y_j = 0$. Thus, consider the remaining case when only one of these 3 terms is nonnegative, say $x_{j1} - x_{j2} - x_{j3} \geq 0$ (similarly for the other terms), then $Y_j = x_{j1} - x_{j2} - x_{j3}$ attains its maximum 1 for $x_{j1} = 1$ and $x_{j2} = x_{j3} = 0$. It follows that if the Boolean formula φ is satisfied by an assignment $x_1, \ldots, x_n \in \{0,1\}$ such that each clause C_j has 1 assigned exactly to one variable, that is, $|\{y_i \in C_j \mid x_i = 1\}| = 1$, then $Y_j = 1$ for every $j \in \{1, \ldots, r\}$ when $(x_1, \ldots, x_n)$ is presented to $\mathcal{N}_\varphi^M$ as an external input. This implies that $\mathcal{N}_\varphi^M(x_1, \ldots, x_n) = r \cdot \frac{M}{r} \cdot 1 = M$ reaches its maximum M at $(x_1, \ldots, x_n)$. Conversely, if $\mathcal{N}_\varphi^M(x_1, \ldots, x_n) \geq M$ for some external input $(x_1, \ldots, x_n) \in [0,1]^n$, then for the same reason, $x_1, \ldots, x_n \in \{0,1\}$ must be an assignment satisfying φ that meets $|\{y_i \in C_j \mid x_i = 1\}| = 1$ for every $j \in \{1, \ldots, r\}$. This completes the proof that the reduction of 1-in-3-SAT+ to MO is correct for the real external input domain $[0,1]^n$. Finally, note that the excitation of the output neuron of $\mathcal{N}_\varphi^M$ is nonnegative according to (18) which proves the NP-hardness of MO also for this case. □

The NP-hardness of MO has negative consequences on the complexity of finding the maximum error (5) of a given NN caused by rounding its weights even for two layers. Namely, given $\mathcal{N}$ with n input neurons, weight-rounding errors δ_{ji} for $j \in V'$ and $i \in j_{\leftarrow}$, from (4), and a positive real constant $M > 0$, then the *maximum weight-rounding error* problem (MWRE) is to decide whether there is an external input $(x_1, \ldots, x_n) \in [0,1]^n$ to $\mathcal{N}$ such that $E(x_1, \ldots, x_n) \geq M$.

Corollary 1. *MWRE is NP-hard for two-layer NNs.*

Proof. We reduce MO with nonnegative output neuron excitations, which is NP-hard according to Theorem 2, to MWRE. For a given MO instance $\mathcal{N}$ with two layers, n input neurons, and one output neuron e (i.e. $Y = \{e\}$) such that $\xi_e \geq 0$, and a real bound $M > 0$, it is sufficient to round only the weights w_{ei} of the output neuron e to $\widetilde{w_{ei}} = 0$ for its inputs $i \in e_{\leftarrow}$, which means $\delta_{ei} = -w_{ei}$ for $i \in e_{\leftarrow}$, whereas $\delta_{ji} = 0$ for $j \in V' \setminus Y$ and $i \in j_{\leftarrow}$. Hence, $\widetilde{\xi_e} = 0$ and

$E(x_1, \ldots, x_n) = |\xi_e| = \xi_e = R(\xi_e) = y_e = \mathcal{N}(x_1, \ldots, x_n)$ from (5) due to $\xi_e \geq 0$, which ensures the correct reduction of MO to MWRE. □

5 Approximating the Maximum Weight-Rounding Error

We have shown in Section 4 that it is computationally hard to compute the maximum of weight-rounding error (5) in the worst case over the whole input domain $[0,1]^n$ of $\mathcal{N}$. Nevertheless, this maximum can simply be approximated by the maximum or average over a finite sample $T \subset [0,1]^n$ which is usually a training or test data chosen according to some input domain distribution, as

$$E_T = \max_{(x_1,\ldots,x_n)\in T} E(x_1, \ldots, x_n)\,, \quad \overline{E_T} = \frac{1}{|T|} \sum_{(x_1,\ldots,x_n)\in T} E(x_1, \ldots, x_n)\,. \quad (19)$$

In this section, we improve this approximation by proposing a method called AppMax that computes the maximum weight-rounding error on a certain convex polytope of $[0,1]^n$ which surrounds a data point from T in the input domain.

To this end, we first introduce the concept of so-called shortcut weights. Given an external input $(x_1, \ldots, x_n) \in [0,1]^n$ to $\mathcal{N}$, some neurons may receive negative excitation and thus remain inactive, i.e., produce zero output. We define

$$S = S(x_1, \ldots, x_n) = \{j \in V' \mid \xi_j < 0\} \quad (20)$$

to be the subset of non-input neurons whose states are *saturated* at $y_j = 0$ according to (2). The complement of this set, $U = V \setminus S$, consists of *unsaturated* units, including both the input neurons $X \subset U$ and any non-input neurons with nonnegative excitation (i.e. $\xi_j \geq 0$ for $j \in V'$). Clearly, fixing a particular pattern of saturation S uniquely determines a corresponding set of active neurons U. Under this fixed activation pattern, $\mathcal{N}$ operates linearly: for each non-input neuron $j \in V'$, the ReLU activation simplifies to either $y_j = 0$ if $j \in S$, or $y_j = \xi_j$ if $j \in U \setminus X$. This turns the entire computation into a linear function of the input values $y_i = x_i$ for $i \in X$. Hence, we can express each excitation ξ_j as:

$$\xi_j = \sum_{i\in X} W_{ji} y_i\,, \quad (21)$$

where $W_{ji} \in \mathbb{R}$ are the so-called *shortcut weights* (including the shortcut bias W_{j0}), capturing the total effect of input neuron $i \in X$ on the excitation of neuron $j \in V'$, under the assumption that the saturation pattern S is fixed. This representation (21) allows us to characterize the saturation condition (20) in terms of shortcut weights:

$$\sum_{i\in X} W_{ji} y_i \begin{cases} < 0 & \text{if } j \in S \\ \geq 0 & \text{if } j \in U\,. \end{cases} \quad (22)$$

Each shortcut weight W_{ji} represents the cumulative influence from input neuron $i \in X$ to unit $j \in V'$ through all unsaturated paths in $\mathcal{N}$. More precisely,

it can be expanded from (1) and (2) as

$$W_{ji} = \sum_{\substack{\text{paths } i=j_0,j_1,\ldots,j_m=j \text{ in } (V,E) \\ j_1,\ldots,j_{m-1}\in U}} \prod_{\ell=1}^{m} w_{j_\ell,j_{\ell-1}} \,, \tag{23}$$

i.e., the sum over all directed paths $i = j_0, j_1, \ldots, j_m = j$ from i to j in the graph (V, E) (i.e. $(j_\ell, j_{\ell-1}) \in E$ for every $\ell \in \{1, \ldots, m\}$) that pass only through unsaturated units $j_1, \ldots, j_{m-1} \in U$, with each path contributing the product $w_{j_1,j_0} w_{j_2,j_1} \cdots w_{j_m,j_{m-1}}$ of edge weights along the path. The shortcut weights can be computed efficiently by propagating the weight values forward through $\mathcal{N}$, in topological order. A formal initialization is performed for input neurons $j \in X$ by:

$$W_{ji} = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{otherwise} \end{cases} \quad \text{for every } i \in X \,. \tag{24}$$

Then, for any non-input neuron $j \in V'$, the shortcut weight is recursively computed as:

$$W_{ji} = \sum_{k \in j_{\leftarrow} \cap U} w_{jk} W_{ki} \quad \text{for each } i \in X \,, \tag{25}$$

assuming the shortcut weights W_{ki} have already been computed for all its predecessors $k \in j_{\leftarrow}$ and $i \in X$.

Furthermore, we construct a NN $\mathcal{N}^*$ that evaluates the weight-rounding error (5) for a given NN $\mathcal{N}$. The NN $\mathcal{N}^*$ is composed of $\mathcal{N}$ and its parallel copy $\widetilde{\mathcal{N}}$ with rounded weights (4), that share the input neurons $X^* = X = \widetilde{X}$ and their originally output units are replaced one by one by new hidden neurons $Y \cup \widetilde{Y} \subset V^*$ in $\mathcal{N}^*$ which have additional connections to the other network with the opposite weights so that

$$\xi_j^* = \xi_j - \widetilde{\xi_j} = \sum_{i \in j_{\leftarrow}} w_{ji} y_i - \sum_{i \in j_{\leftarrow}} \widetilde{w_{ji}}\, \widetilde{y_i} \quad \text{for } j \in Y \tag{26}$$

$$\xi_j^* = \widetilde{\xi_j} - \xi_j = \sum_{i \in j_{\leftarrow}} \widetilde{w_{ji}}\, \widetilde{y_i} - \sum_{i \in j_{\leftarrow}} w_{ji} y_i \quad \text{for } j \in \widetilde{Y} \,. \tag{27}$$

In addition, we add one new output unit e in $\mathcal{N}^*$ (i.e. $Y^* = \{e\}$) on the top of these new neurons with the unit weights and zero bias, which computes

$$\begin{aligned} \mathcal{N}^*(x_1, \ldots, x_n) = y_e^* = \xi_e^* &= \sum_{j \in Y} y_j^* + \sum_{j \in \widetilde{Y}} y_j^* = \sum_{j \in Y} R\left(\xi_j^*\right) + \sum_{j \in \widetilde{Y}} R\left(\xi_j^*\right) \\ &= \sum_{j \in Y} \left(R\left(\xi_j - \widetilde{\xi_j}\right) + R\left(\widetilde{\xi_j} - \xi_j\right) \right) = \sum_{j \in Y} \left| \xi_j - \widetilde{\xi_j} \right| = E(x_1, \ldots, x_n) \end{aligned} \tag{28}$$

according to (2), (26), (27), and (5).

Finally, we calculate the shortcut weights W_{ji} of units $j \in V^*$ for input neurons $i \in X$ of $\mathcal{N}^*$, according to (24) and (25). Thus, for any fixed set $S \subset V^*$

of saturated units in $\mathcal{N}^*$, which induces $U = V^* \setminus S$, we can formulate a linear program of finding the states $y_1, \ldots, y_n$ of its input neurons $X \setminus \{0\}$ that

$$\text{maximize} \quad W_{e0} + \sum_{i=1}^{n} W_{ei} y_i \tag{29}$$

$$\text{subject to} \quad W_{j0} + \sum_{i=1}^{n} W_{ji} y_i \le 0 \quad \text{for every } j \in S \tag{30}$$

$$W_{j0} + \sum_{i=1}^{n} W_{ji} y_i \ge 0 \quad \text{for every } j \in U \setminus X \tag{31}$$

$$0 \le y_i \le 1 \quad \text{for every } i \in \{1, \ldots, n\}\,. \tag{32}$$

By solving this program we obtain the maximum of weight-rounding error (28) for the original $\mathcal{N}$ on the convex polytope $\Xi_S \subseteq [0,1]^n$ in its input domain, that is defined by (30)–(32) for a given $S \subset V^*$ according to (22) where zero excitations are allowed for saturated units without loss of generality:

$$E_{\Xi_S} = \max_{(y_1,\ldots,y_n)\in\Xi_S} E(y_1, \ldots, y_n)\,. \tag{33}$$

Hence, in our AppMax method, we can improve the maximum error approximation (19) as

$$E_{\Xi_{S(T)}} = \max_{(x_1,\ldots,x_n)\in T} E_{\Xi_{S(x_1,\ldots,x_n)}}\,, \quad \overline{E_{\Xi_{S(T)}}} = \frac{1}{|T|} \sum_{(x_1,\ldots,x_n)\in T} E_{\Xi_{S(x_1,\ldots,x_n)}} \tag{34}$$

which takes the maximum or average not only over data points $(x_1, \ldots, x_n)$ from T but over the convex polytopes $\Xi_{S(x_1,\ldots,x_n)}$ around these data points.

For classification tasks, the regression error (5) under the L_1 norm can be replaced by the cross-entropy between the output categorical probability distributions (3) of $\mathcal{N}$ and $\widetilde{\mathcal{N}}$, respectively,

$$L(x_1, \ldots, x_n) = -\sum_{k\in Y} y_k \ln \widetilde{y_k} = \sum_{k\in Y} y_k \ln \left(1 + \sum_{j\in\widetilde{Y}\setminus\{k\}} e^{\widetilde{\xi_j} - \widetilde{\xi_k}}\right)\,. \tag{35}$$

We interpret the output from $\mathcal{N}$ to be the class with the maximum excitation of output neurons which discretizes the softmax probabilities (3) by the winner-take-all principle. For each input $(x_1, \ldots, x_n) \in T$, the correct class inferred by the trained $\mathcal{N}$ is thus $c = \arg\max_{k\in Y} \xi_k$, which means that $y_c \ge 1/m$ by (3), where $m = |Y|$ is the number of classes. Hence, we can lower bound the cross-entropy loss (35), subject to $\xi_c \ge \xi_k$ for all $k \in Y \setminus \{c\}$, as

$$L(x_1, \ldots, x_n) \ge \frac{1}{m} \ln \left(1 + \exp\left(\max_{j\in\widetilde{Y}\setminus\{c\}} \left(\widetilde{\xi_j} - \widetilde{\xi_c}\right)\right)\right)\,. \tag{36}$$

In this case, contrary to (26)–(27), the original unmodified units from $Y \cup \widetilde{Y}$ create the output neurons of $\mathcal{N}^*$ instead of the removed e. The convex polytope

$\Xi_S \subseteq [0,1]^n$ in (33) is then defined by (30)–(32) for the saturated units $S = S(x_1,\ldots,x_n) \subset V^* \setminus (Y \cup \widetilde{Y})$ in this new $\mathcal{N}^*$, inducing $U = V^* \setminus (S \cup Y \cup \widetilde{Y})$, which is further constrained by conditions:

$$W_{c0} + \sum_{i=1}^{n} W_{ci} y_i \geq W_{k0} + \sum_{i=1}^{n} W_{ki} y_i \quad \text{for every } k \in Y \setminus \{c\}\,. \tag{37}$$

We replace the argmax of (35) on a feasible convex set $\Xi_{S(x_1,\ldots,x_n)}$ for $(x_1,\ldots,x_n) \in T$, that meets $\xi_c \geq \xi_k$ for all $k \in Y \setminus \{c\}$, due to (37), by the argument of

$$\max_{j \in \widetilde{Y} \setminus \{c\}} \max_{\substack{(y_1,\ldots,y_n) \in \Xi_{S(x_1,\ldots,x_n)} \\ \widetilde{\xi}_c \leq \widetilde{\xi}_j}} (\widetilde{\xi}_j - \widetilde{\xi}_c)\,, \tag{38}$$

according to (36), which is computed by solving the $|Y|-1$ linear programs that maximize $\widetilde{\xi}_j - \widetilde{\xi}_c$ subject to (30)–(32), (37), and $\widetilde{W_{c0}} + \sum_{i=1}^{n} \widetilde{W_{ci}} y_i \leq \widetilde{W_{j0}} + \sum_{i=1}^{n} \widetilde{W_{ji}} y_i$, for each $j \in \widetilde{Y} \setminus \{c\}$ separately. Clearly, if the nonnegative error $\widetilde{\xi}_j - \widetilde{\xi}_c \geq 0$ is feasible, then $\widetilde{\mathcal{N}}$ does not infer the correct class produced by $\mathcal{N}$.

Note that the proposed AppMax method of computing the maximum error can be used for any approximation of $\mathcal{N}$ by $\widetilde{\mathcal{N}}$, which can be an arbitrary DNN with the same number of input and output neurons, e.g. created by pruning hidden neurons and connections in $\mathcal{N}$.

6 Experiments

We have tested the AppMax method introduced in Section 5 on two NNs trained on the MNIST database [7] of handwritten digits (28x28 grayscale pixels) categorized into 10 classes (0–9). We carried out our experiments using the deep learning library PyTorch [21] and the SciPy linear programming routine `scipy.optimize.linprog` [29, 10]. The source code is publicly available [28].

The first Net1 is a NN of three fully connected layers with 784–2000–1000–10 neurons (including $784 = 28^2$ input units), respectively. The second Net2 is a CNN composed of two convolutional layers with 32 and 64 3x3-kernels (stride 1, padding 1), respectively, followed by one max pooling layer with 64 2x2-kernels (stride 2), and topped by two fully connected layers with 1024–10 neurons, respectively. The convolutional and max-pooling layers in Net2 are transformed to eight fully connected layers with 784–25088–50176–50176–25088–25088–12544–1024–10 neurons (see Section 2) where sparse matrices are used to fit the weights into the memory. Both NNs employ the ReLU activation function (2) and were

Table 2. Accuracy on the test set.

	32b	16b	12b	8b	6b	4b
Net1	98.30	98.30	98.30	98.30	98.30	24.85
Net2	99.25	99.25	99.25	99.25	99.25	99.14

Table 3. The worst-case maximum error estimates E_T and $E_{\Xi_{S(T)}}$ and their averaged variants $\overline{E_T}$ and $\overline{E_{\Xi_{S(T)}}}$ for the weight rounding to 16 bits.

	E_T	$E_{\Xi_{S(T)}}$	$\overline{E_T}$	$\overline{E_{\Xi_{S(T)}}}$
Net1	0.032854	0.099374	0.007629	0.030884
Net2	0.013466	0.014763	0.006127	0.006777

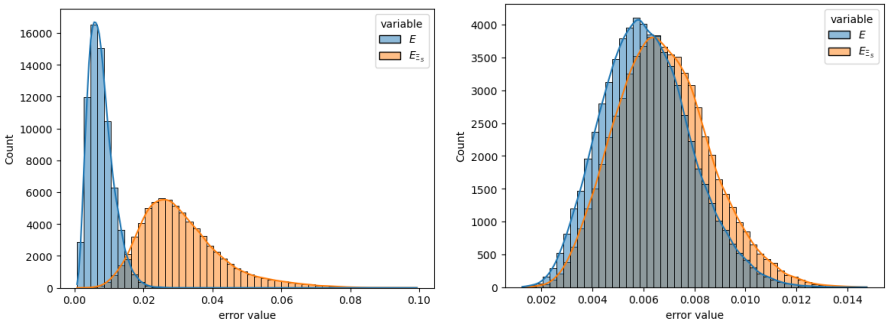

Fig. 1. The histograms of weight-rounding errors E on T (blue) and E_{Ξ_S} over convex polytopes Ξ_S on T (orange) according to (19) and (33), respectively, for Net1 (left) and Net2 (right) with 16-bitwidth for weights.

originally trained with 32-bitwidth for weights. Tab. 2 lists accuracies of Net1 and Net2 on the test set (i.e. the percentage of correctly classified data points) for weights that are rounded to 16, 12, 8, 6, and 4 bits, showing their high robustness in terms of accuracy only.

We applied the AppMax method to approximate the maximum of regression error (5) under L_1 norm for Net1 and Net2 whose weights are rounded to 16 bits. All available (training and test) 70,000 data points were first used as a sample T in the maximum error estimates (19) and (34), where the latter thus involves solving 70,000 linear programs (29)–(32). The results of this experiment are listed in Tab. 3 and graphically depicted using histograms in Fig. 1. We can observe a clear shift in the weight-rounding errors E on T and E_{Ξ_S} over convex polytopes Ξ_S on T according to (19) and (33), respectively. This shift is less pronounced in Net2 due to its higher number of neurons, which results in smaller convex polytopes. This effect could be mitigated by also considering the convex polytopes adjacent to those containing the test data points, thereby expanding the region in which the maximum error is searched for. The AppMax method clearly improves the estimation of the maximum weight-rounding error. For illustration, the worst-case points found by the AppMax method (33) in corresponding convex polytopes around several data points are depicted in Fig. 2. They include noise, which is quite visible for Net1, while negligible for Net2.

The AppMax method, which is based on solving large linear programs for each data point in the sample T, turns out to be computationally intensive. For example, the presented experiments performed on 70,000 data points required

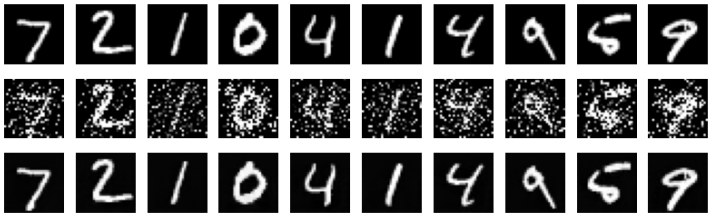

Fig. 2. The first line shows a few examples of original data points $(x_1, \ldots, x_n) \in T$, whereas the second and third lines depict the points $(y_1, \ldots, y_n) \in \Xi_{S(x_1,\ldots,x_n)}$ with the maximum error $E_{\Xi_{S(x_1,\ldots,x_n)}}$ found by the AppMax method in the convex polytopes $\Xi_{S(x_1,\ldots,x_n)}$ for Net1 and Net2, respectively, with 16-bitwidth for weights.

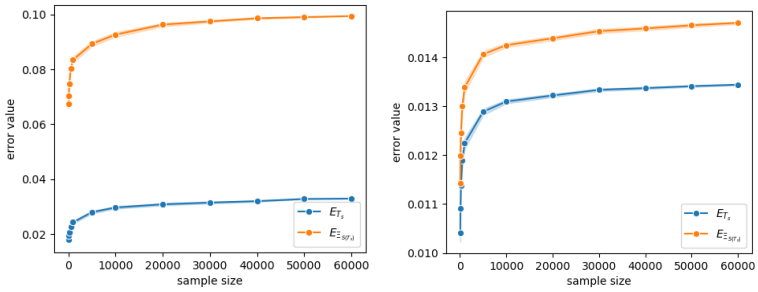

Fig. 3. Estimates of errors E_{T_s} (blue) and $E_{\Xi_{S(T_s)}}$ (orange) for increasing size of data samples T_s for Net1 (left) and Net2 (right) with 16-bitwidth for weights.

several days using tens of parallel processors. Specifically, the average computation time per one data point was approximately 8 seconds for Net1 and 250 seconds for Net2 on an Intel® Xeon® E5-2620 v4 processor running at 2.10 GHz. In the next experiment, we try to answer the question, how many data points do we need to get reliable estimate of $E_{\Xi_{S(T)}}$. We sampled randomly sets $T_s \subset T$ from the available data points, ranging from 50 to 60,000 sample size, repeating each sample hundred times. For each sample, the approximate maximum errors E_{T_s} and $E_{\Xi_{S(T_s)}}$ were evaluated according to (19) and (34), respectively, as depicted in Fig. 3. The results show that it is not necessary to employ the whole data to get reasonable error estimates because the improvement using tens of thousands data points is not significant. This brings the required computation time down to a reasonable level—on the order of several hours in our setting.

Furthermore, we demonstrate the effect of decreasing bitwidth on the weight-rounding errors (19) and (34). We used 10000 and 2000 randomly chosen test data points as a sample T for Net1 and Net2, respectively, due to the larger Net2 is slower to evaluate. We summarize the results only for Net1 in Tab. 4, as Net2 shows similar trends. Moreover, Fig. 4 depicts graphically the increase of error E_{Ξ_S} (on a logarithmic scale) for decreasing bitwidth of weights in Net1. Clearly, the AppMax method improves the error guarantees for different bitwidths.

Table 4. The error estimates E_T and $E_{\Xi_{S(T)}}$ and their averaged variants $\overline{E_T}$ and $\overline{E_{\Xi_{S(T)}}}$ for Net1 with decreasing bitwidth of weights.

	E_T	$E_{\Xi_{S(T)}}$	$\overline{E_T}$	$\overline{E_{\Xi_{S(T)}}}$
16 bits	0.024727	0.093156	0.007558	0.030998
12 bits	0.613171	1.049668	0.135616	0.384750
8 bits	8.191886	17.585771	2.138221	6.070758
6 bits	40.410836	85.562221	10.226672	25.475516
4 bits	301.230476	479.39271	81.117751	153.583925

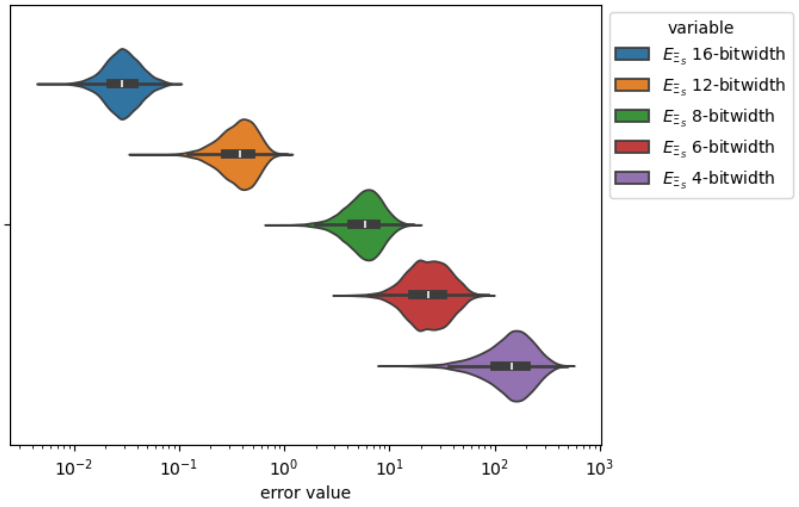

Fig. 4. The violin plots showing E_{Ξ_S} for different bitwidths of weights in Net1.

Finally, we test the AppMax variant (38) for the classification by Net1 with decreasing bitwidth of weights on a sample T of 5000 randomly chosen data points. For each $(x_1, \ldots, x_n) \in T$ which is classified by Net1 as a digit class $c \in Y$, the AppMax method found the argument of (38) in the convex polytope $\Xi_{S(x_1,\ldots,x_n)}$ (by solving 9 linear programs) whose all points are classified by Net1 as the same digit class c due to (37), if it exists. This argmax is a point misclassified by weight-rounded Net1 as another digit class $g \in Y \setminus \{c\}$. Tab. 5 lists the percentage of data points in T that are surrounded by convex polytopes containing misclassified points (shortly, misclassified polytopes), which seems to be not affected by the bitwidth for weights. The next columns of Tab. 5 present the output probabilities $\overline{\widetilde{y_c}}$ and $\overline{\widetilde{y_g}}$ of correct and wrong classes, respectively, averaged over T (restricted to data points surrounded by misclassified polytopes) that were provided by Net1 with rounded weights using the softmax function (3) for the argument of (38) as its input, where $c, g \in Y$ may differ for each of these points. In Tab. 5, these are further compared to the analogous averaged output probabilities $\overline{y_c}$ and $\overline{y_g}$ of correct and wrong classes, respectively, by the original Net1

Table 5. The percentage of data points in T that are surrounded by misclassified polytopes including the arguments of (38), for which the output probabilities $\overline{\widetilde{y_c}}$ and $\overline{\widetilde{y_g}}$ of the correct and wrong classes, respectively, averaged over these points, are listed, which were provided by Net1 with decreasing bitwidth for weights, and compared to those by the original Net1.

	percentage of	rounded Net1		original Net1	
bitwidth	misclassified polytopes	$\overline{\widetilde{y_c}}$	$\overline{\widetilde{y_g}}$	$\overline{y_c}$	$\overline{y_g}$
16 bits	0.91	0.28	0.28	0.28	0.28
12 bits	0.91	0.27	0.30	0.29	0.29
8 bits	0.91	0.15	0.39	0.31	0.30
6 bits	0.88	0.14	0.51	0.40	0.32
4 bits	0.89	0.08	0.44	0.57	0.16

for the same inputs. It turns out that the AppMax method reveals the increase of the cross-entropy restricted to the correct and wrong classes for decreasing bitwidth of weights although the percentage of misclassified polytopes surrounding the data points from T seems to be not affected by the weight rounding.

7 Conclusion

In this paper, we theoretically analyzed the effect of any weight rounding in a trained DNN on its output. We derived an upper bound on the worst-case output regression error for bounded input domains, which appears to be overestimated for practical use. We proved that it is in fact NP-hard to determine this error precisely. We introduced the AppMax method that approximates the weight-rounding error by computing its maxima in convex polytopes around the data points. This method can be used for any DNN approximation. We tested the AppMax method on fully connected and convolutional NNs trained on the MNIST database for decreasing bitwidth of weights, showing a clear improvement in error guarantees as compared to those for test data points only. The proposed AppMax method enables systematic comparison of approximation strategies to identify those with optimal accuracy–performance trade-offs, paving the way for techniques that reduce energy consumption by approximating or removing suitable DNN components with reliable error guarantees.

The AppMax method has only been partially generalized to classification based on the softmax function. In future research, we plan to explore the possibility of approximating the cross-entropy loss using a piecewise linear function or addressing the underlying nonlinear optimization problem using techniques such as the Karush-Kuhn-Tucker conditions. Another open problem concerns approximating the weight-rounding error on a global scale by estimating the probabilities of considered convex polytopes in terms of their volumes. We also plan to extend the error analysis to modern architectures such as ResNet, Transformers. One of the most important challenges is to identify components in a given DNN that can be neglected (e.g. specific weights to be rounded) at the cost of an explicitly bounded increase in the output error. We will also extend our

experiments with the AppMax method to other test datasets including CIFAR-100 [13] and ImageNet [5].

Acknowledgments. This research was institutionally supported by RVO: 67985807, J. Šíma was partially supported by the Czech Science Foundation grant GA25-15490S, and P. Vidnerová was partially supported by the Strategy AV21 project "AI: Artificial Intelligence for Science and Society."

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Ansari, M.S., et al.: Improving the accuracy and hardware efficiency of neural networks using approximate multipliers. IEEE Trans. Very Large Scale Integr. VLSI Syst. **28**, 317–328 (2020). https://doi.org/10.1109/TVLSI.2019.2940943
2. Armeniakos, G., et al.: Hardware approximate techniques for deep neural network accelerators: A survey. ACM Comput. Surv. **55**, 83 (2023). https://doi.org/10.1145/3527156
3. Beuzeville, T., Buttari, A., Gratton, S., Mary, T.: Deterministic and probabilistic backward error analysis of neural networks in floating-point arithmetic. Manuscript (2024). `https://hal.science/hal-04663142`
4. Chen, Y., et al.: Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE J. Solid-State Circuits **52**, 127–138 (2017). https://doi.org/10.1109/JSSC.2016.2616357
5. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A large-scale hierarchical image database. In: Proc. CVPR 2009. pp. 248–255 (2009). https://doi.org/10.1109/CVPR.2009.5206848
6. Deng, L., et al.: Model compression and hardware acceleration for neural networks: A comprehensive survey. Proc. IEEE **108**, 485–532 (2020). https://doi.org/10.1109/JPROC.2020.2976475
7. Deng, L.: The MNIST database of handwritten digit images for machine learning research. IEEE Signal Process Mag. **29**, 141–142 (2012). https://doi.org/10.1109/MSP.2012.2211477
8. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. In: Proc. ICLR 2016 (2016). https://doi.org/10.48550/arXiv.1510.00149
9. Horowitz, M.: 1.1 Computing's energy problem (and what we can do about it). In: Proc. ISSCC 2014. pp. 10–14 (2014). https://doi.org/10.1109/ISSCC.2014.6757323
10. Huangfu, Q., Hall, J.A.J.: Parallelizing the dual revised simplex method. Math. Program. Comput. **10**, 119–142 (2018). https://doi.org/10.1007/S12532-017-0130-5
11. Jouppi, N.P., et al.: A domain-specific architecture for deep neural networks. Commun. ACM **61**, 50–59 (2018). https://doi.org/10.1145/3154484
12. Kim, Y., et al.: Compression of deep convolutional neural networks for fast and low power mobile applications. In: Proc. ICLR 2016 (2016). https://doi.org/10.48550/arXiv.1511.06530
13. Krizhevsky, A., Nair, V., Hinton, G.: CIFAR (Canadian Institute for Advanced Research), `http://www.cs.toronto.edu/~kriz/cifar.html`

14. Lee, E.H., et al.: LogNet: Energy-efficient neural networks using logarithmic computation. In: Proc. ICASSP 2017. pp. 5900–5904 (2017). https://doi.org/10.1109/ICASSP.2017.7953288
15. Li, Z., Li, H., Meng, L.: Model compression for deep neural networks: A survey. Computers **12**, 60 (2023). https://doi.org/10.3390/COMPUTERS12030060
16. Lyu, Z., et al.: A survey of model compression strategies for object detection. Multimedia Tools Appl. **83**, 48165–48236 (2024). https://doi.org/10.1007/S11042-023-17192-X
17. Mittal, S.: A survey of techniques for approximate computing. ACM Comput. Surv. **48**, 62 (2016). https://doi.org/10.1145/2893356
18. Mittal, S.: A survey of FPGA-based accelerators for convolutional neural networks. Neural Comput. Appl. **32**, 1109–1139 (2020). https://doi.org/10.1007/S00521-018-3761-1
19. Mittal, S., et al.: A survey of SRAM-based in-memory computing techniques and applications. J. Syst. Archit. **119**, 102276 (2021). https://doi.org/10.1016/J.SYSARC.2021.102276
20. Moons, B., Verhelst, M.: An energy-efficient precision-scalable ConvNet processor in 40-nm CMOS. IEEE J. Solid-State Circuits **52**, 903–914 (2017). https://doi.org/10.1109/JSSC.2016.2636225
21. Paszke, A., et al.: PyTorch: An imperative style, high-performance deep learning library. CoRR, arXiv:1912.01703 [cs.LG] (2019). https://doi.org/10.48550/arXiv.1912.01703
22. Schaefer, T.J.: The complexity of satisfiability problems. In: Proc. STOC 1978. pp. 216–226 (1978). https://doi.org/10.1145/800133.804350
23. Silvano, C., et al.: A survey on deep learning hardware accelerators for heterogeneous HPC platforms. CoRR, arXiv:2306.15552 [cs.AR] (2023). https://doi.org/10.48550/arXiv.2306.15552
24. Šíma, J., Cabessa, J., Vidnerová, P.: On energy complexity of fully-connected layers. Neural Netw. **178**, 106419 (2024). https://doi.org/10.1016/J.NEUNET.2024.106419
25. Šíma, J., Vidnerová, P., Mrázek, V.: Energy complexity of convolutional neural networks. Neural Comput. **36**, 1601–1625 (2024). https://doi.org/10.1162/NECO_A_01676
26. Sze, V., et al.: Efficient Processing of Deep Neural Networks. Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers (2020). https://doi.org/10.2200/S01004ED1V01Y202004CAC050
27. Tang, Y., et al.: A survey on transformer compression. CoRR, arXiv:2402.05964 [cs.LG] (2024). https://doi.org/10.48550/arXiv.2402.05964
28. Vidnerová, P.: Source code for experiments. (2024). `https://github.com/PetraVidnerova/RoundingErrorEstimation`
29. Virtanen, P., et al.: SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nat. Methods **17**, 261–272 (2020). https://doi.org/10.1038/s41592-019-0686-2
30. Wang, P., Cheng, J.: Fixed-point factorized networks. In: Proc. CVPR 2017. pp. 3966–3974 (2017). https://doi.org/10.1109/CVPR.2017.422
31. Yang, T., Chen, Y., Sze, V.: Designing energy-efficient convolutional neural networks using energy-aware pruning. In: Proc. CVPR 2017. pp. 6071–6079 (2017). https://doi.org/10.1109/CVPR.2017.643
32. Zhou, G., Zhou, J., Lin, H.: Research on NVIDIA deep learning accelerator. In: Proc. ASID 2018. pp. 192–195 (2018). https://doi.org/10.1109/ICASID.2018.8693202