Cross-Entropy Loss of Approximated 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, Czechia
{sima,petra}@cs.cas.cz

Abstract. Deep neural networks (DNNs), which underpin modern AI technologies, demand substantial computational resources, posing challenges for deployment on energy-constrained devices (e.g., battery-powered smartphones). A viable solution is to reduce the complexity of trained DNNs via approximate computing techniques, such as low-bit quantization or pruning, which significantly lower energy consumption with minimal impact on inference accuracy. In this paper, we adapt our AppMax method—originally developed for estimating regression error of approximated neural networks (NNs)—to upper-bound the cross-entropy loss between the output categorical probability distributions of a trained classification DNN with softmax (e.g., a convolutional NN) and its low-energy approximation. Using the concept of shortcut weights and optimal linear interpolation of the exponential function, AppMax bounds this loss via linear programming over convex polytopes around test/training data points, constrained to regions where the same category is originally inferred with high probability. Preliminary MNIST experiments show that AppMax identifies inputs with maximum cross-entropy loss, some of which are misclassified by the approximated NN (with reduced weight bitwidth), even though its overall accuracy on the test data is preserved. This error bound can be used to evaluate different approximation strategies and identify those that best balance accuracy and energy efficiency.

Keywords: Deep neural networks · Approximate Computing · Cross-Entropy Loss · Linear programming.

1 Introduction

Deep neural networks (DNNs) have become central to modern artificial intelligence, powering applications such as image and speech recognition, natural language processing, and robotics. Their growing deployment in embedded systems—ranging from autonomous surveillance to smart healthcare—brings advanced functionality to resource-constrained devices. However, the high computational and energy demands of DNNs, often composed of numerous layers and millions of parameters, present a major hurdle for on-device inference, particularly in mobile and wearable technologies where battery life is critical.

To address these challenges, research has focused on improving the energy efficiency of DNN processing through two main complementary strategies [24].

The first involves hardware-level optimization: specialized DNN accelerators [9, 20], GPUs [30], FPGAs [16], and in-memory computing platforms [17] exploit massive parallelism to reduce computational overhead while adhering to principled energy complexity bounds [23, 21]. The second strategy, aimed at error-resilient applications, employs approximate computing to trade slight reductions in accuracy for substantial energy savings [2, 4, 13–15, 25]. Techniques include model compression [3], pruning [29], compact network architectures [10], reduced-precision arithmetic, approximate multipliers [1], and quantization schemes—such as fixed-point representation [28], bitwidth reduction [18], nonuniform quantization [12], weight sharing [6]—which have been shown to dramatically lower energy consumption per operation [7].

It has been observed that energy consumption during DNN inference is dominated not only by numerical computation but also by data movement, with memory access often accounting for over 70% of the total cost [29]. Reducing the complexity of approximated DNNs can significantly mitigate both of these energy components. In our previous work [22], we theoretically analyzed the effect of arbitrary weight rounding—specified by individual deviations—on the output of a trained DNN employing the ReLU (rectified linear unit) activation function, where the approximation may thus be generated by any method, such as those referenced above (e.g., reduced bitwidth, quantization). We derived a global worst-case upper bound on the DNN regression error (under the L_1 norm) induced by a given weight rounding, which turns out to be overestimated for practical tasks. Moreover, we proved that tightening this upper bound is NP-hard, even for two-layer networks.

Therefore, we developed a method called AppMax, which computes the maximum L_1 regression error of an approximated DNN using linear programming, at least over convex polytopes surrounding test/training data points [22]. To this end, we introduced the concept of so-called shortcut weights—coefficients expressing the linear dependence of DNN outputs on inputs under a fixed saturation of neuron states, enabled by the piecewise linearity of ReLU (Section 2).

In this paper, we adapt the AppMax method to neural networks (NNs) trained for categorical classification employing a softmax output layer (Section 4), encompassing convolutional neural networks (CNNs) with (max) pooling layers. AppMax thus provides an upper bound on the cross-entropy loss between the output categorical probability distributions of the original classification DNN and its low-energy DNN approximation, evaluated over the convex polytopes, constrained to regions in which the same category is originally inferred with high probability (Section 3). To achieve this, we linearly interpolate the exponential function at optimal points that minimize the approximation error (Appendix).

In this preliminary study, we test the AppMax method on a small, fully connected NN trained on the MNIST dataset, which was approximated by reducing the bitwidth of its weights (Section 5). The experiments demonstrate that our AppMax method yields more reliable estimates of the worst-case error than those computed on the test data points alone for which the accuracy of the approximated NN can be perfectly preserved. Moreover, AppMax identifies inputs with maximum cross-entropy loss over the convex polytopes surrounding the data points, some of which are misclassified by the approximated NN.

2 A Formal Model of DNNs and Shortcut Weights

We define a formal model of (artificial) feedforward neural networks (NNs) with the ReLU (rectified linear unit) activation function, which encompasses deep neural networks (DNNs) commonly used for classification tasks, such as convolutional neural networks (CNNs). The architecture of a NN $\mathcal{N}$ is a connected, directed acyclic graph (V, E) where $E \subset V \times V$ and V is composed of units, called neurons, whose real-valued states (outputs) are denoted by 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 n-dimensional real-valued input to $\mathcal{N}$ from a bounded domain, say $(x_1, \ldots, x_n) \in [0,1]^n$ without loss of generality [22], that is $y_j = x_j$ for $j \in X$. Moreover, a set of $m \geq 2$ output neurons, $Y \subseteq V' = V \setminus X$ provides the output $\mathcal{N}(x_1, \ldots, x_n) \in [0,1]^m$ from $\mathcal{N}$ for this input, which is interpreted as m categorical probabilities. 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-valued bias, which, as usual, can be viewed as the 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 $w_{j0} \neq 0$. 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 state y_j of a non-output neuron $j \in V' \setminus Y$ is computed by applying the ReLU activation function $R : \mathbb{R} \to \mathbb{R}$ to its excitation ξ_j,

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

The states y_j of output neurons $j \in Y$ normalize their excitations into a categorical probability distribution by using the softmax function

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

The convolutional layers in CNNs are sometimes interleaved with max pooling layers whose units $j \in V' \setminus Y$ 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). 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.). Similarly, average pooling can be implemented. Thus, we will hereafter assume without loss of generality that $\mathcal{N}$ does not contain pooling layers.

Furthermore, we introduce the concept of so-called *shortcut weights* [22]. For any external input $(x_1,\ldots,x_n) \in [0,1]^n$ to $\mathcal{N}$, we define a subset of hidden neurons (i.e. excluding input and output units),

$$S = S(x_1,\ldots,x_n) = \{j \in V' \setminus Y \mid \xi_j < 0\} \tag{4}$$

whose states are said to be *saturated* at $y_j = 0$ according to (2), whereas its complement of *unsaturated* units is denoted by $U = V \setminus (S \cup Y)$, which, formally, also includes the input neurons $X \subseteq U$. Under such a fixed saturation, the excitation ξ_j of neuron $j \in V'$ is a linear function of the states $y_1,\ldots,y_n$ of input neurons (i.e. any external input to $\mathcal{N}$) that satisfy (4), since only the linear parts of the ReLU function are used, namely, either $y_j = 0$ if $j \in S$ or $y_j = \xi_j$ if $j \in U \setminus X$, according to (2). Hence, for each neuron $j \in V'$, we define its real-valued shortcut weights $W_{ji} \in \mathbb{R}$ for input neurons $i \in X$ (including its shortcut bias W_{j0}), which are coefficients of this linear function, $\xi_j = \sum_{i \in X} W_{ji} y_i$ for any external input that meets (4). If condition (4) is satisfied, then it can be rewritten using the shortcut weights as

$$\sum_{i \in X} W_{ji} y_i \begin{cases} < 0 & \text{if } j \in S \\ \geq 0 & \text{if } j \in U \end{cases} \quad \text{for every } j \in V' \setminus Y\,. \tag{5}$$

For any unit $j \in V'$, the shortcut weight W_{ji} for input neuron $i \in X$ can be expanded from (1) and (2) as

$$W_{ji} = \sum_{\substack{\text{path } i=j_0,j_1,\ldots,j_m=j \text{ in } (V,E) \\ j_1,\ldots,j_{m-1} \in U}} \prod_{\ell=1}^{m} w_{j_\ell, j_{\ell-1}} \tag{6}$$

which can be efficiently computed using feedforward propagation through $\mathcal{N}$. We start with the input neurons $j \in X \subseteq U$, for which we formally define $W_{ji} = 1$ if $j = i$, whereas $W_{ji} = 0$ otherwise, for every $i \in X$. Then for any non-input unit $j \in V'$, we calculate its shortcut weights as

$$W_{ji} = \sum_{k \in j_{\leftarrow} \cap U} w_{jk} W_{ki} \quad \text{for every } i \in X\,, \tag{7}$$

provided that the shortcut weights W_{ki} have already been computed for all neurons $k \in j_{\leftarrow}$ and $i \in X$.

3 Upper Bounding the Cross-Entropy Loss

Consider a NN $\mathcal{N}$ that is approximated by another NN $\widetilde{\mathcal{N}}$ (e.g., by rounding the weights to a given number of binary digits in their floating-point representations), which shares the same inputs, $\widetilde{X} = X$, and has the same number of outputs, that is, $|\widetilde{Y}| = |Y| = m$. We denote the corresponding categorical probabilities and excitations of output neurons $k \in \widetilde{Y}$ by $\widetilde{y_k}$ and $\widetilde{\xi_k}$, respectively. For any input $(y_1, \ldots, y_n) \in [0,1]^n$, we can measure the error of $\widetilde{\mathcal{N}}$ using the cross-entropy loss between the categorical probability distributions of $\mathcal{N}$ and $\widetilde{\mathcal{N}}$:

$$L(y_1, \ldots, y_n) = -\sum_{k \in Y} y_k \ln \widetilde{y_k} \,. \tag{8}$$

which upper bounds the Kullback-Leibler divergence of $\mathcal{N}$ from $\widetilde{\mathcal{N}}$.

Suppose a data point $(x_1, \ldots, x_n) \in T$ from a test or training set $T \subseteq [0,1]^n$ belongs to the category $c \in Y$. We aim to find an upper bound on the cross-entropy loss (8) over a neighborhood $\varXi_S \cap \varXi_{\widetilde{S}}$ of this point, where

$$\varXi_S = \{(y_1, \ldots, y_n) \in [0,1]^n \mid S(y_1, \ldots, y_n) = S\} \tag{9}$$

is a convex polytope defined by the set of saturated units $S = S(x_1, \ldots, x_n)$ for the input $(x_1, \ldots, x_n)$ presented to $\mathcal{N}$ according to (5), and $\varXi_{\widetilde{S}}$ is specified analogously for $\widetilde{\mathcal{N}}$. In addition, this neighborhood is further constrained by the condition

$$y_c \geq p \tag{10}$$

for some probability threshold p (e.g., $p = 0.8$), that is,

$$\varXi = \left\{(y_1, \ldots, y_n) \in \varXi_S \cap \varXi_{\widetilde{S}} \mid y_c \geq p\right\} \tag{11}$$

which includes only those inputs that $\mathcal{N}$ classifies into the same category $c \in Y$ as the data point $(x_1, \ldots, x_n)$ with probability at least p. Using the softmax function (3), the constraint (10) can be rewritten as

$$\sum_{j \in Y \setminus \{c\}} e^{\xi_j - \xi_c} \leq \frac{1-p}{p} \,. \tag{12}$$

From the cross-entropy loss (8), we isolate the term corresponding to the output $c \in Y$ and bound the remaining sum from above using the summand with maximum $\ln(1/\widetilde{y_k}) > 0$:

$$L(y_1, \ldots, y_n) \leq y_c \ln \frac{1}{\widetilde{y_c}} + (1 - y_c) \max_{k \in \widetilde{Y} \setminus \{c\}} \ln \frac{1}{\widetilde{y_k}} \tag{13}$$

since $0 < y_k < 1$ for all $k \in Y$, and $\sum_{k \in Y} y_k = 1$ due to (3). Because the logarithmic function is increasing, this can be rewritten by plugging the softmax function (3) as

$$\begin{aligned} L(y_1, \ldots, y_n) &\leq \max_{k \in \widetilde{Y} \setminus \{c\}} \left(\ln \frac{1}{\widetilde{y_c}^{\,y_c}} + \ln \frac{1}{\widetilde{y_k}^{\,1-y_c}} \right) = \ln \left(\max_{k \in \widetilde{Y} \setminus \{c\}} \frac{1}{\widetilde{y_k}} \left(\frac{\widetilde{y_k}}{\widetilde{y_c}} \right)^{y_c} \right) \\ &= \ln \left(\max_{k \in \widetilde{Y} \setminus \{c\}} \sum_{j \in \widetilde{Y}} e^{\widetilde{\xi_j} - \widetilde{\xi_k} + y_c \left(\widetilde{\xi_k} - \widetilde{\xi_c} \right)} \right) \end{aligned} \tag{14}$$

which can further be upper bounded for $(y_1, \ldots, y_n) \in \Xi$ as

$$L(y_1, \ldots, y_n) \leq \ln \left(\max_{k \in \widetilde{Y} \setminus \{c\}} \sum_{j \in \widetilde{Y}} e^{\widetilde{\xi_j} - \widetilde{\xi_k} + R\left(\widetilde{\xi_k} - \widetilde{\xi_c}\right) - pR\left(\widetilde{\xi_c} - \widetilde{\xi_k}\right)} \right) \tag{15}$$

according to (2), since $p \leq y_c < 1$ holds in Ξ due to (11).

Notice that both the constraint formula (12) and the loss function bound (15) involve the exponential function applied to linear functions from (2) and (1), depending on the excitations of the output neurons of $\mathcal{N}$ and $\widetilde{\mathcal{N}}$, respectively. In order to evaluate these formulas using a NN, we linearly interpolate the exponential function e^x at $r + 3$ points $a_0 < a_1 < \cdots < a_n < a_{r+1} < a_{r+2}$ by a continuous piecewise linear function $\mathcal{N}_e(x)$.

For $i \in \{0, \ldots, r + 1\}$, denote by m_i and b_i the slope and y-intercept, respectively, of the linear interpolant between a_i and a_{i+1}. This means $z_i = e^{a_i} = m_i a_i + b_i$ and $z_{i+1} = e^{a_{i+1}} = m_i a_{i+1} + b_i$ which determines

$$m_i = \frac{e^{a_{i+1}} - e^{a_i}}{a_{i+1} - a_i} \quad \text{and} \quad b_i = \frac{a_{i+1} e^{a_i} - a_i e^{a_{i+1}}}{a_{i+1} - a_i} \quad \text{for } i \in \{0, \ldots, r + 1\}\,. \tag{16}$$

In the Appendix, we show how to find the r unique points $a_1, \ldots, a_r$ inside a given interval $[a_0, a_{r+1}]$ that minimize the approximation error

$$E(a_1, \ldots, a_r) = \sum_{i=0}^{r} \int_{a_i}^{a_{i+1}} (m_i x + b_i - e^x)\, dx\,, \tag{17}$$

which prove to satisfy

$$e^{a_i} = \frac{e^{a_{i+1}} - e^{a_{i-1}}}{a_{i+1} - a_{i-1}} \quad \text{for } i \in \{1, \ldots, r\}\,. \tag{18}$$

In addition, a_{r+2} is chosen to be sufficiently large so that it exceeds the common arguments of the exponential function in (15) but not so large that the exponential values become numerically unstable in computer calculations (e.g., due to floating-point overflow), which further constrains the polytope Ξ.

The interpolation $\mathcal{N}_e(x)$ of e^x extends to any real x as

$$\mathcal{N}_e(x) = \begin{cases} z_0 & \text{if } x \leq a_0 \\ m_q x + b_q & \text{if } a_q < x \leq a_{q+1} \text{ for some } q \in \{0, \ldots, r + 1\} \\ m_{r+1} x + b_{r+1} & \text{if } x > a_{r+2}\,, \end{cases} \tag{19}$$

which satisfies $e^x \leq \mathcal{N}_e(x)$ for $x \in (-\infty, a_{r+2}]$ and $e^{a_i} = \mathcal{N}_e(a_i)$ for $i \in \{0, \ldots, r + 2\}$, since e^x is a convex function. Hence, $\mathcal{N}_e$ can be used to rewrite the constraint (12) as

$$\sum_{j \in Y \setminus \{c\}} e^{\xi_j - \xi_c} \leq \sum_{j \in Y \setminus \{c\}} \mathcal{N}_e(\xi_j - \xi_c) \leq \frac{1 - p}{p} \tag{20}$$

for $a_{r+2} \geq \ln(1-p)/p \geq a_0$ (e.g., $a_{r+2} \geq 0$ and $a_0 = -5$ suffice for $0.5 \leq p \leq 0.99$), and the loss function bound (15) as

$$\ln\left(\max_{k\in\widetilde{Y}\setminus\{c\}}\max_{(y_1,\ldots,y_n)\in\overline{\Xi}}\sum_{j\in\widetilde{Y}}\mathcal{N}_e\left(\widetilde{\xi}_j-\widetilde{\xi}_k+R\left(\widetilde{\xi}_k-\widetilde{\xi}_c\right)-pR\left(\widetilde{\xi}_c-\widetilde{\xi}_k\right)\right)\right) \quad (21)$$

for all $(z_1,\ldots,z_n)\in\overline{\Xi}$ such that $\widetilde{\xi}_j-\widetilde{\xi}_k+R\left(\widetilde{\xi}_k-\widetilde{\xi}_c\right)-pR\left(\widetilde{\xi}_c-\widetilde{\xi}_k\right)\leq a_{r+2}$, where $\overline{\Xi}$ is the closure of Ξ including zero excitations for saturated units.

4 The AppMax Method

In this section, we will adapt our AppMax method [22] to compute an upper bound on the cross-entropy loss between the categorical probability distributions of $\mathcal{N}$ and $\widetilde{\mathcal{N}}$ over a convex polytope surrounding a data point $(x_1,\ldots,x_n)\in T$ which belongs to category $c\in Y$. For $c\in Y$ and each $k\in\widetilde{Y}\setminus\{c\}$, we construct a NN $\mathcal{N}^*_{ck}$ that evaluates the sums

$$\sum_{j\in Y\setminus\{c\}}\mathcal{N}_e(\xi_j-\xi_c) \quad\text{and}\quad \sum_{j\in\widetilde{Y}}\mathcal{N}_e\left(\widetilde{\xi}_j-\widetilde{\xi}_k+R\left(\widetilde{\xi}_k-\widetilde{\xi}_c\right)-pR\left(\widetilde{\xi}_c-\widetilde{\xi}_k\right)\right) \quad (22)$$

for an input $(y_1,\ldots,y_n)\in\overline{\Xi}$, which occur in the constraint (20) and the loss function bound (21), respectively. The NN $\mathcal{N}^*_{ck}$ is composed of $\mathcal{N}$ and $\widetilde{\mathcal{N}}$ placed side-by-side in parallel, which share the same input neurons $X^*=X=\widetilde{X}$. Moreover, their output neurons Y and $\widetilde{Y}$ are replaced by subnetworks $\mathcal{N}^*_c$ and $\widetilde{\mathcal{N}}^*_k$ that evaluate the sums (22) using neurons based on equations (1) and (2).

We prove that the interpolation formula (19) can be implemented by a two-layer NN using $2r+3$ hidden neurons as

$$\mathcal{N}_e(x)=z_0+\sum_{i=0}^{r+1}R(m_ix+b_i-z_i)-\sum_{i=0}^{r}R(m_ix+b_i-z_{i+1})\,. \quad (23)$$

For all $i\in\{0,\ldots,r+1\}$, we have $m_ix+b_i-z_{i+1}<m_ix+b_i-z_i\leq 0$ iff $x\leq a_i$, by the definition of z_i, and $m_i>0$ due to (16). Suppose $a_q<x\leq a_{q+1}$ for some $q\in\{0,\ldots,r+1\}$. Then $R(m_ix+b_i-z_i)=m_ix+b_i-z_i>0$ if $0\leq i\leq q$ whereas $R(m_ix+b_i-z_i)=0$ if $q+1\leq i\leq r+1$. Hence, $\mathcal{N}_e(x)=z_0+\sum_{i=0}^{q}(m_ix+b_i-z_i)-\sum_{i=0}^{q-1}(m_ix+b_i-z_{i+1})=m_qx+b_q$. For $x\leq a_0$, formula (23) gives $\mathcal{N}_e(x)=z_0$, whereas $\mathcal{N}_e(x)=m_{r+1}x+b_{r+1}$ for $x>a_{r+2}$, which completes the proof that (23) implements (19).

In order to implement $\mathcal{N}^*_c$, we plug (23) into the first sum in (22) and obtain $\sum_{j\in Y\setminus\{c\}}\mathcal{N}_e(\xi_j-\xi_c)=\sum_{j\in Y\setminus\{c\}}\big(z_0+\sum_{i=0}^{r+1}R\left(m_i(\xi_j-\xi_c)+b_i-z_i\right)$

$-\sum_{i=0}^{r} R\left(m_i(\xi_j - \xi_c) + b_i - z_{i+1}\right)\big)$ which reduces to the formula

$$\xi_o^* = (m-1)z_0 + \sum_{j \in Y \setminus \{c\}} \left(\sum_{i=0}^{r+1} R\left(\sum_{\ell \in j_{\leftarrow}} m_i w_{j\ell} y_\ell - \sum_{\ell \in c_{\leftarrow}} m_i w_{c\ell} y_\ell + b_i - z_i \right) \right.$$
$$\left. - \sum_{i=0}^{r} R\left(\sum_{\ell \in j_{\leftarrow}} m_i w_{j\ell} y_\ell - \sum_{\ell \in c_{\leftarrow}} m_i w_{c\ell} y_\ell + b_i - z_{i+1} \right) \right) \quad (24)$$

that depends on the states of non-output neurons $\ell \in V \setminus Y$ in $\mathcal{N}$, according to (1). Thus, the subnetwork $\mathcal{N}_c^*$ evaluates formula (24) in the excitation $\xi_o^* = \sum_{j \in Y \setminus \{c\}} \mathcal{N}_e(\xi_j - \xi_c)$ of its output neuron $o \in Y^*$ using two layers and $2r+3$ hidden units.

Similarly, the subnetwork $\widetilde{\mathcal{N}}_k^*$ computes the second sum in (22) in the excitation $\xi_{\widetilde{o}}^*$ of its output neuron $\widetilde{o} \in Y^*$ as $\xi_{\widetilde{o}}^* = mz_0 + \sum_{j \in \widetilde{Y}} \big(\sum_{i=0}^{r+1} R(m_i \widetilde{\zeta}_{ckj}^* + b_i - z_i)$ $- \sum_{i=0}^{r} R(m_i \widetilde{\zeta}_{ckj}^* + b_i - z_{i+1})\big)$ including the subexcitations

$$\widetilde{\zeta}_{ckj}^* = \sum_{\ell \in j_{\leftarrow}} \widetilde{w}_{j\ell} \widetilde{y}_\ell - \sum_{\ell \in k_{\leftarrow}} \widetilde{w}_{k\ell} \widetilde{y}_\ell + R\left(\sum_{\ell \in k_{\leftarrow}} \widetilde{w}_{k\ell} \widetilde{y}_\ell - \sum_{\ell \in c_{\leftarrow}} \widetilde{w}_{c\ell} \widetilde{y}_\ell \right)$$
$$- p\,R\left(\sum_{\ell \in c_{\leftarrow}} \widetilde{w}_{c\ell} \widetilde{y}_\ell - \sum_{\ell \in k_{\leftarrow}} \widetilde{w}_{k\ell} \widetilde{y}_\ell \right) \qquad \text{for } j \in \widetilde{Y}\,, \quad (25)$$

which is implemented using three layers with $2m$ and $2r+3$ hidden neurons, respectively. Altogether, the size of $\mathcal{N}_{ck}^*$ is $|V^*| = |V \setminus Y| + |\widetilde{V'} \setminus \widetilde{Y}| + 2m + 4r + 8$ units, including $n = |X^*|$ input and $|Y^*| = |\{o, \widetilde{o}\}| = 2$ output neurons.

Let $S^* = S_{ck}^*(x_1, \ldots, x_n)$ be the set of saturated units in $\mathcal{N}_{ck}^*$ for the external input $(x_1, \ldots, x_n)$ according to (4), which induces the set $U^* = V^* \setminus (S^* \cup Y^*)$ of unsaturated neurons. For this fixed saturation, we calculate the shortcut weights W_{ji} of units $j \in V^*$ in $\mathcal{N}_{ck}^*$ for its input neurons $i \in X^*$, according to (7). Analogously, we determine the shortcut weights V_{ji} for the subexcitations $\widetilde{\zeta}_{ckj}^* = \sum_{i \in X^*} V_{ji} y_i$ for $j \in \widetilde{Y}$. Thus, we can formulate a linear program of finding the states $y_1, \ldots, y_n$ of its input neurons $X^* \setminus \{0\}$ that

$$\text{maximize} \quad \xi_{\widetilde{o}}^* = W_{\widetilde{o}0} + \sum_{i=1}^{n} W_{\widetilde{o}i} y_i \quad (26)$$

$$\text{subject to} \quad \xi_o^* = W_{o0} + \sum_{i=1}^{n} W_{oi} y_i \le \frac{1-p}{p} \quad (27)$$

$$\widetilde{\zeta}_{ckj}^* = V_{j0} + \sum_{i=1}^{n} V_{ji} y_i \le a_{r+2} \quad \text{for every } j \in \widetilde{Y} \quad (28)$$

$$\xi_j^* = W_{j0} + \sum_{i=1}^{n} W_{ji} y_i \le 0 \quad \text{for every } j \in S^* \quad (29)$$

$$\xi_j^* = W_{j0} + \sum_{i=1}^{n} W_{ji} y_i \geq 0 \quad \text{for every } j \in U^* \setminus X^* \tag{30}$$

$$0 \leq y_i \leq 1 \quad \text{for every } i \in \{1, \ldots, n\} . \tag{31}$$

For each $k \in \widetilde{Y} \setminus \{c\}$, we solve the linear program (26)–(31) to determine the maximum

$$\sigma_k = \max_{(y_1,\ldots,y_n) \in \Xi_{ck}^*} \xi_o^* = \max_{(y_1,\ldots,y_n) \in \Xi_{ck}^*} \sum_{j \in \widetilde{Y}} \mathcal{N}_e \left(\widetilde{\xi}_j - \widetilde{\xi}_k + R \left(\widetilde{\xi}_k - \widetilde{\xi}_c \right) \right. \\ \left. - pR \left(\widetilde{\xi}_c - \widetilde{\xi}_k \right) \right) \tag{32}$$

where $\Xi_{ck}^* \subseteq \overline{\Xi}$ is a closed convex polytope around the data point $(x_1, \ldots, x_n)$, consisting of points $(y_1, \ldots, y_n)$ that satisfy the constraints (27)–(31), according to (11). From (21) and (32), it follows that the cross-entropy loss can then be upper bounded as

$$L(y_1, \ldots, y_n) \leq \ln \left(\max_{k \in \widetilde{Y} \setminus \{c\}} \sigma_k \right) \tag{33}$$

for every input $(y_1, \ldots, y_n) \in \bigcup_{k \in \widetilde{Y} \setminus \{c\}} \Xi_{ck}^*$.

5 Experiments

For a preliminary study, we tested our AppMax method, introduced in Section 4, on a simple, small NN $\mathcal{N}$. Nevertheless, AppMax had previously demonstrated its computational efficiency even on large CNNs when estimating regression error [22]. Thus, $\mathcal{N}$ is composed of three fully connected layers with 784–64–32–10 neurons (including $n = 784$ input and $m = 10$ output units), respectively, which employs the ReLU activation function (2) in the two hidden layers and the softmax function (3) in the output layer. It was trained with 32-bitwidth for weights on the MNIST database [5] of handwritten digits (28x28 grayscale pixels) categorized into 10 classes (0–9). We carried out our experiments using the deep learning library PyTorch [19] and the SciPy linear programming routine `scipy.optimize.linprog` [27, 8]. The source code is publicly available [26].

The approximated NN $\widetilde{\mathcal{N}}$ was derived from $\mathcal{N}$ by rounding its weights to 4 binary digits in their floating-point representations. In the experiments we use test data corresponding to one class c, particularly the digit 1, which comprises 1135 data points. The accuracy of $\mathcal{N}$ and $\widetilde{\mathcal{N}}$ on this test set is 100% and 99.91%, respectively. Specifically, $\widetilde{\mathcal{N}}$ classifies correctly (i.e. $\text{argmax}_{k \in Y} \widetilde{y}_k = c$) all but one data point, which shows high robustness with respect to the data points. In the AppMax method we use the linear interpolation of the exponential function (see Section 3) on the interval $[-5, 5]$ (i.e. $a_0 = -5$ and $a_{r+1} = 5$) at $r = 14$ optimal internal points (see the Appendix) extended to sufficiently large $a_{r+2} = 20$.

We apply the AppMax method to find inputs within the convex polytope (27)–(31) that maximize the upper bound (21) on the cross-entropy loss between

Table 1. Number of misclassified polytopes out of 1135 around the test data points.

p	0.30	0.40	0.50	0.60	0.70	0.80	0.90	0.95	0.99
misclassified	134	71	30	11	8	0	0	0	0

the categorical probability distributions of $\mathcal{N}$ and $\widetilde{\mathcal{N}}$. This is done for different values of the probability threshold p with which at least the corresponding class c is inferred by $\mathcal{N}$ according to (10) implemented by (27). Table 1 shows the number of the so-called misclassified polytopes around the original test data points in which these inputs with maximum cross-entropy loss are classified by $\widetilde{\mathcal{N}}$ into a category $\kappa = \text{argmax}_{k\in Y}\, \widetilde{y}_k$ different from the category c inferred by $\mathcal{N}$ (we call them misclassified inputs). Clearly, the number of misclassified polytopes decreases when these are more constrained by increasing the probability threshold p.

Furthermore, Fig. 1 depicts the histograms of output probabilities y_c and y_κ for these misclassified inputs generated for different p (top down) which are compared for $\mathcal{N}$ (left) and $\widetilde{\mathcal{N}}$ (right). The diagrams show that for the misclassified inputs, the approximated $\widetilde{\mathcal{N}}$ produces the categorical probability distributions with widely spread low y_c and high y_κ, which clearly differ from those by the original $\mathcal{N}$ separating high y_c from low y_κ. Thus, the AppMax method identifies the data points belonging to the category c, in the vicinity of which there are inputs that are assigned to the same category c by $\mathcal{N}$ with probability at least p while they are misclassified by $\widetilde{\mathcal{N}}$. This exposes the weaknesses of the approximated $\widetilde{\mathcal{N}}$ even though its accuracy on the test set is almost the same as that of the original $\mathcal{N}$.

6 Conclusion

In this paper, we have modified our AppMax technique [22] to estimate an upper bound on the cross-entropy loss between the categorical probability distributions of a trained classification DNN and its corresponding low-energy DNN approximation. We tested experimentally the method on a simple fully connected NN trained on the MNIST dataset, which is approximated by reducing the bitwidth of the weight representations. Our results demonstrate a notable refinement of the error bounds compared to results derived from test data alone.

In future studies, we aim to tackle the non-linear optimization challenge of locating the maximum cross-entropy loss, potentially using techniques such as the Karush-Kuhn-Tucker conditions. Another open question involves approximating this error on a global scale by estimating the volumes of relevant convex polytopes to infer probability distributions. A key objective remains leveraging this error analysis to pinpoint parts of a DNN—such as specific weights—that can be simplified (e.g., rounded) while ensuring a provably bounded increase in output error. We also intend to expand our preliminary experimental validation of the AppMax method to larger CNNs (already used in its previous tests [22]), and to other benchmark datasets, including CIFAR-10 [11].

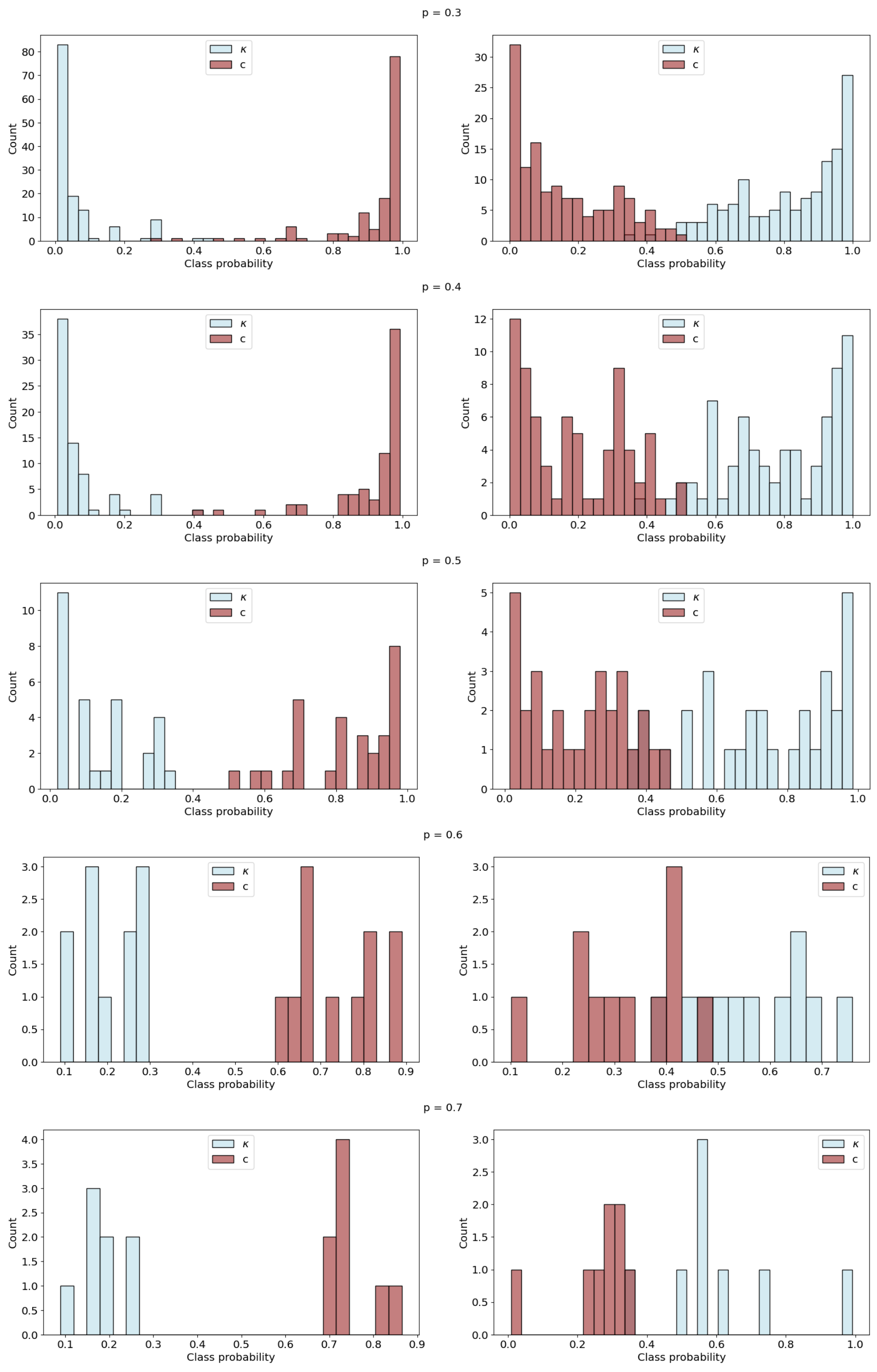

Fig. 1. Histograms of output probabilities y_c and y_κ over misclassified inputs (with maximum cross-entropy loss estimated by our AppMax method in misclassified polytopes around data points from the category c) that are classified into the category c by $\mathcal{N}$ (left) with the probability at least p and into category $\kappa \neq c$ by $\widetilde{\mathcal{N}}$ (right).

Acknowledgments. This research was institutionally supported by RVO: 67985807; J. Šíma was partially supported by the Czech Science Foundation grant GA25-15490S; P. Vidnerová was partially supported by the research programme of the Strategy AV21 "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.

Appendix: The Optimal Interpolation Points

We show how to compute the optimal interpolation points $a_1, \ldots, a_r$ inside a given interval $[a_0, a_{r+1}]$ for the exponential function e^x so as to minimize the approximation error (17). Note that $\varepsilon_i(x) = m_i x + b_i - e^x \geq 0$ for all $x \in [a_i, a_{i+1}]$ and $i \in \{0, \ldots, r\}$, due to the convexity of the exponential function. The antiderivative $\mathcal{E}_i(x) = \frac{m_i}{2} x^2 + b_i x - e^x$ of $\varepsilon_i(x)$ is used to calculate the error (17) as

$$\begin{aligned} E(a_1, \ldots, a_r) &= \sum_{i=0}^{r} \left(\mathcal{E}_i(a_{i+1}) - \mathcal{E}_i(a_i)\right) \\ &= \sum_{i=0}^{r} \left(\frac{m_i}{2}\left(a_{i+1}^2 - a_i^2\right) + b_i\left(a_{i+1} - a_i\right) - e^{a_{i+1}} + e^{a_i}\right) \end{aligned} \tag{34}$$

according to the fundamental theorem of calculus. Substituting (16) into (34), we obtain

$$\begin{aligned} E(a_1, \ldots, a_r) &= \sum_{i=0}^{r} \left(\tfrac{1}{2}\left(e^{a_{i+1}} - e^{a_i}\right)\left(a_{i+1} + a_i\right) + a_{i+1} e^{a_i} - a_i e^{a_{i+1}} - e^{a_{i+1}} + e^{a_i}\right) \\ &= \sum_{i=0}^{r} \left(\tfrac{1}{2}\left(a_{i+1} e^{a_{i+1}} - a_i e^{a_i} + a_{i+1} e^{a_i} - a_i e^{a_{i+1}}\right) - e^{a_{i+1}} + e^{a_i}\right) \end{aligned} \tag{35}$$

which sums up to

$$\begin{aligned} E(a_1, \ldots, a_r) &= \tfrac{1}{2}\left(a_{r+1} e^{a_{r+1}} - a_0 e^{a_0}\right) + \tfrac{1}{2} \sum_{i=0}^{r} \left(a_{i+1} e^{a_i} - a_i e^{a_{i+1}}\right) - e^{a_{r+1}} + e^{a_0} \\ &= \tfrac{1}{2} \sum_{i=1}^{r} \left(a_{i+1} - a_{i-1}\right) e^{a_i} + \left(\frac{a_1 - a_0}{2} + 1\right) e^{a_0} \\ &\quad + \left(\frac{a_{r+1} - a_r}{2} - 1\right) e^{a_{r+1}}. \end{aligned} \tag{36}$$

We set the partial derivatives equal to zero:

$$\frac{\partial E}{\partial a_i} = \tfrac{1}{2}\left(a_{i+1} - a_{i-1}\right) e^{a_i} - \tfrac{1}{2}\left(e^{a_{i+1}} - e^{a_{i-1}}\right) = 0 \quad \text{for } i \in \{1, \ldots, r\}\,, \tag{37}$$

which provides the necessary conditions (18) for $(a_1, \dots, a_r) \in \mathbb{R}^r$ to be a minimizer of $E(a_1, \dots, a_r)$. We apply the second partial derivative test to show that the conditions (18) are also sufficient. The second partial derivatives

$$h_{ij} = \frac{\partial E}{\partial a_i \partial a_j} = \begin{cases} \frac{1}{2}\left(a_{i+1} - a_{i-1}\right) e^{a_i} & \text{if } j = i \\ \frac{1}{2}\left(e^{a_{i-1}} - e^{a_i}\right) & \text{if } j = i-1 \\ \frac{1}{2}\left(e^{a_i} - e^{a_{i+1}}\right) & \text{if } j = i+1 \\ 0 & \text{otherwise} \end{cases} \quad \text{for } i, j \in \{1, \dots, r\}\,, \tag{38}$$

form the entries of the $r \times r$ Hessian matrix $\mathbf{H}_E = (h_{ij})$. Denote by $\mathbf{H}$ this Hessian $\mathbf{H}_E$ at $a_1, \dots, a_r$ satisfying (18) which means the diagonal entries of $\mathbf{H}$ in (38) read

$$h_{ii} = \tfrac{1}{2}\left(e^{a_{i+1}} - e^{a_{i-1}}\right) \quad \text{for } i \in \{1, \dots, r\}\,. \tag{39}$$

We prove that $\mathbf{H}$ is positive definite. For any nonzero real vector $\mathbf{x}^\top = (x_1, \dots, x_r) \in \mathbb{R}^r \setminus \{\mathbf{0}\}$, we have

$$\mathbf{x}^\top \mathbf{H}\,\mathbf{x} = \frac{1}{2}\sum_{i=1}^{r}\left(e^{a_{i+1}} - e^{a_{i-1}}\right) x_i^2 - \sum_{i=1}^{r-1}\left(e^{a_{i+1}} - e^{a_i}\right) x_i x_{i+1} \tag{40}$$

according to (38) and (39). By subtracting and adding e^{a_i} to $e^{a_{i+1}} - e^{a_{i-1}}$, the first sum in (40) can be split into two sums as

$$\sum_{i=1}^{r}\left(e^{a_{i+1}} - e^{a_{i-1}}\right) x_i^2 = \sum_{i=1}^{r}\left(e^{a_{i+1}} - e^{a_i}\right) x_i^2 + \sum_{i=0}^{r-1}\left(e^{a_{i+1}} - e^{a_i}\right) x_{i+1}^2 \tag{41}$$

which being plugged into (40), gives

$$\begin{aligned} \mathbf{x}^\top \mathbf{H}\,\mathbf{x} = \frac{1}{2}&\sum_{i=1}^{r-1}\left(e^{a_{i+1}} - e^{a_i}\right)\left(x_i - x_{i+1}\right)^2 \\ &+ \left(e^{a_1} - e^{a_0}\right) x_1^2 + \left(e^{a_{r+1}} - e^{a_r}\right) x_r^2 > 0\,. \end{aligned} \tag{42}$$

Hence, the conditions (18) determine a local minimum of $E(a_1, \dots, a_r)$.

In addition, observe that $(a_1, \dots, a_r)$ satisfying the conditions (18), would represent the unique global minimum of $E(a_1, \dots, a_r)$. Given a_0 and a_1, these conditions determine uniquely element by element the subsequent sequence $a_2, \dots, a_r, a_{r+1}$ which ends with the given fixed point a_{r+1}. This is because, geometrically, a_{i+1} is the x-coordinate of the intersection $e^{a_{i+1}} = m_i a_{i+1} + b_i$ of the exponential function and the line drawn from the point $(a_{i-1}, e^{a_{i-1}})$ with slope $m_i = e^{a_i}$ for each $i \in \{1, \dots, r\}$, according to (18). Thus, for different values of a_1 we could not achieve the same a_{r+1}.

This global minimum can be approximated using the following fixed-point iteration, which also proves its existence. We can start with any ordered distinct points $a_1^{(0)}, \dots, a_r^{(0)}$ in the open interval (a_0, a_{r+1}), for example, equally spaced

$$a_i^{(0)} = a_0 + \frac{a_{r+1} - a_0}{r+1}\, i \quad \text{for } i \in \{1, \dots, r\}\,, \tag{43}$$

and continue

$$a_i^{(t+1)} = \begin{cases} \ln\left(\frac{e^{a_{i+1}^{(t)}} - e^{a_{i-1}^{(t)}}}{a_{i+1}^{(t)} - a_{i-1}^{(t)}}\right) & \text{if } t = kr + i \\ a_i^{(t)} & \text{otherwise} \end{cases} \quad \text{for } i \in \{1, \dots, r\}\,, \;\; k \geq 0 \tag{44}$$

where $a_0^{(t)} = a_0$ and $a_{r+1}^{(t)} = a_{r+1}$ for all $t \geq 0$, according to (18). It follows from (44) and (36) that

$$\begin{aligned} \Delta_{t+1} &= E\left(a_1^{(t+1)}, \dots, a_r^{(t+1)}\right) - E\left(a_1^{(t)}, \dots, a_r^{(t)}\right) \\ &= \tfrac{1}{2}\left(a_{i+1}^{(t)} - a_{i-1}^{(t)}\right)\left(e^{a_i^{(t+1)}} - e^{a_i^{(t)}}\right) - \tfrac{1}{2}\left(e^{a_{i+1}^{(t)}} - e^{a_{i-1}^{(t)}}\right)\left(a_i^{(t+1)} - a_i^{(t)}\right) \end{aligned} \tag{45}$$

for any $t = kr + i \geq 0$ such that $i \in \{1, \dots, r\}$ and $k \geq 0$. Clearly, $\Delta_{t+1} = 0$ if $a_i^{(t+1)} = a_i^{(t)}$.

For $a_i^{(t+1)} \neq a_i^{(t)}$, we show that $\Delta_{t+1} < 0$ which is equivalent to

$$\frac{e^{a_i^{(t+1)}} - e^{a_i^{(t)}}}{a_i^{(t+1)} - a_i^{(t)}} \begin{cases} < e^{a_i^{(t+1)}} & \text{if } a_i^{(t+1)} > a_i^{(t)} \\ > e^{a_i^{(t+1)}} & \text{if } a_i^{(t+1)} < a_i^{(t)} \end{cases} \tag{46}$$

due to (45) and (44). Consider the case when $a_i^{(t+1)} > a_i^{(t)}$, whereas the argument for $a_i^{(t+1)} < a_i^{(t)}$ is analogous. By the mean value theorem for the exponential function on the closed interval $\left[a_i^{(t)}, a_i^{(t+1)}\right]$, there is $a \in \left(a_i^{(t)}, a_i^{(t+1)}\right)$ such that

$$e^a = \frac{e^{a_i^{(t+1)}} - e^{a_i^{(t)}}}{a_i^{(t+1)} - a_i^{(t)}} \tag{47}$$

which implies (46) because $e^a < e^{a_i^{(t+1)}}$, completing the proof of $\Delta_{t+1} < 0$ for $a_i^{(t+1)} \neq a_i^{(t)}$. Since $E(a_1, \dots, a_r)$ is a bounded function, this ensures that the fixed-point iteration (44) converges to its global minimum $a_i = \lim_{t\to\infty} a_i^{(t)}$ for $i \in \{1, \dots, r\}$, which satisfies (18).

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)
2. Armeniakos, G., et al.: Hardware approximate techniques for deep neural network accelerators: A survey. ACM Comput. Surv. **55**, 83 (2023)
3. Chen, Y., et al.: Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE J. Solid-State Circuits **52**, 127–138 (2017)
4. Deng, L., et al.: Model compression and hardware acceleration for neural networks: A comprehensive survey. Proc. IEEE **108**, 485–532 (2020)
5. Deng, L.: The MNIST database of handwritten digit images for machine learning research. IEEE Signal Process Mag. **29**, 141–142 (2012)

6. 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)
7. Horowitz, M.: 1.1 Computing's energy problem (and what we can do about it). In: Proc. ISSCC 2014. pp. 10–14 (2014)
8. Huangfu, Q., Hall, J.A.J.: Parallelizing the dual revised simplex method. Math. Program. Comput. **10**, 119–142 (2018)
9. Jouppi, N.P., et al.: A domain-specific architecture for deep neural networks. Commun. ACM **61**, 50–59 (2018)
10. Kim, Y., et al.: Compression of deep convolutional neural networks for fast and low power mobile applications. In: Proc. ICLR 2016 (2016)
11. Krizhevsky, A., Nair, V., Hinton, G.: CIFAR (Canadian Institute for Advanced Research), http://www.cs.toronto.edu/ kriz/cifar.html
12. Lee, E.H., et al.: LogNet: Energy-efficient neural networks using logarithmic computation. In: Proc. ICASSP 2017. pp. 5900–5904 (2017)
13. Li, Z., Li, H., Meng, L.: Model compression for deep neural networks: A survey. Computers **12**, 60 (2023)
14. Lyu, Z., et al.: A survey of model compression strategies for object detection. Multimedia Tools Appl. **83**, 48165–48236 (2024)
15. Mittal, S.: A survey of techniques for approximate computing. ACM Comput. Surv. **48**, 62 (2016)
16. Mittal, S.: A survey of FPGA-based accelerators for convolutional neural networks. Neural Comput. Appl. **32**, 1109–1139 (2020)
17. Mittal, S., et al.: A survey of SRAM-based in-memory computing techniques and applications. J. Syst. Archit. **119**, 102276 (2021)
18. Moons, B., Verhelst, M.: An energy-efficient precision-scalable ConvNet processor in 40-nm CMOS. IEEE J. Solid-State Circuits **52**, 903–914 (2017)
19. Paszke, A., et al.: PyTorch: An imperative style, high-performance deep learning library. CoRR, arXiv:1912.01703 [cs.LG] (2019)
20. Silvano, C., et al.: A survey on deep learning hardware accelerators for heterogeneous HPC platforms. CoRR, arXiv:2306.15552 [cs.AR] (2023)
21. Šíma, J., Cabessa, J., Vidnerová, P.: On energy complexity of fully-connected layers. Neural Netw. **178**, 106419 (2024)
22. Šíma, J., Vidnerová, P.: Weight-rounding error in deep neural networks (to appear at ECML PKDD 2025), https://www.cs.cas.cz/sima/rnderr.pdf
23. Šíma, J., Vidnerová, P., Mrázek, V.: Energy complexity of convolutional neural networks. Neural Comput. **36**, 1601–1625 (2024)
24. Sze, V., et al.: Efficient Processing of Deep Neural Networks. Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers (2020)
25. Tang, Y., et al.: A survey on transformer compression. CoRR, arXiv:2402.05964 [cs.LG] (2024)
26. Vidnerová, P.: Source code for experiments, https://github.com/PetraVidnerova/ClassificationRoundingErrors
27. Virtanen, P., et al.: SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nat. Methods **17**, 261–272 (2020)
28. Wang, P., Cheng, J.: Fixed-point factorized networks. In: Proc. CVPR 2017. pp. 3966–3974 (2017)
29. Yang, T., Chen, Y., Sze, V.: Designing energy-efficient convolutional neural networks using energy-aware pruning. In: Proc. CVPR 2017. pp. 6071–6079 (2017)
30. Zhou, G., Zhou, J., Lin, H.: Research on NVIDIA deep learning accelerator. In: Proc. ASID 2018. pp. 192–195 (2018)