
# Weight-Rounding Error in Deep Neural Networks

**Jiří Šíma**
sima@cs.cas.cz

*joint work with* **Petra Vidnerová**
petra@cs.cas.cz

Institute of Computer Science
Czech Academy of Sciences, Prague, Czechia

# Efficient Processing of Deep Neural Networks (DNNs)

- DNNs are widely used in many artificial intelligence applications (e.g. large language models, image recognition, computer vision, robotics, etc.)
- achieve state-of-the-art accuracy, but with high computational complexity (often tens of millions of operations for a single inference)
- the energy efficiency of DNN implementations on low-power, battery-operated hardware (e.g. cellphones, smartwatches, smart glasses) becomes crucial

$\longrightarrow$ **reducing the energy cost of DNNs:**

(`Sze,Chen,Yang,Emer:Efficient Processing of Deep Neural Networks,2020`)

**1. Hardware Design:** energy efficient implementation of DNNs on various hardware platforms, including GPUs, FPGAs, in-memory computing architectures

$\approx$ 70% of energy is consumed on data movement within the memory hierarchy, with the rest on numerical computations

a hardware-independent model of energy complexity for DNNs unifies asymptotic lower and upper bounds on energy consumption across diverse DNN accelerators (`Šíma,Vidnerová,Mrázek,2024`)

**2. Approximate Computing** methods in error-tolerant applications (e.g. image classification) save large amounts of energy with minimal accuracy loss by reducing

- model size: pruning, compression, weight sharing, approximate multipliers
- arithmetic precision: fixed-point operations, reduction of weight bit-width, nonuniform quantization

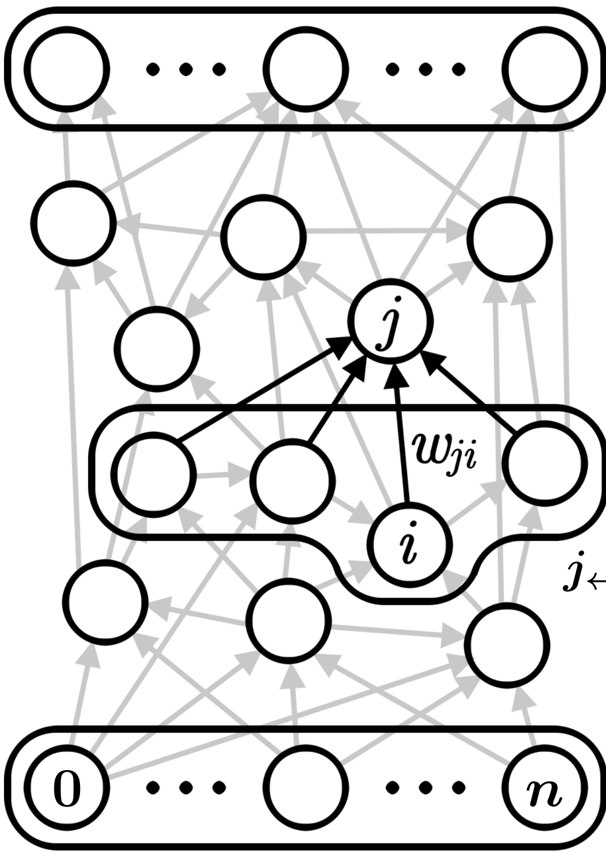
**Example:** an 8-bit fixed-point multiply consumes $18.5\times$ less energy than a 32-bit floating-point multiply (Horowitz,2014), corresponding to additional fourfold energy reduction for data memory transfers—the most energy-intensive operation

**The aim of this study:** theoretical analysis of the effect of (post-training) weight rounding on DNN output to guarantee maximum error bounds

- rounding is specified by individual weight deviations, which can be generated by any method, such as reduced bitwidth or quantization etc.
- here, we consider the regression error of approximated DNNs, measured under $L_1$ norm, later generalized to cross-entropy loss for classification tasks (Šíma,Vidnerová,ICONIP 2025)
- our main results apply to any approximated DNNs, including those obtained, for example, via pruning

# Formal Model of DNNs

the architecture of a DNN $\mathcal{N}$ is a connected directed acyclic graph $(V, E)$ composed of neurons, where edges $(i, j) \in E \subset V \times V$ are labeled with weights $w_{ji} \in \mathbb{R}$




$Y \subseteq V$ output neurons

$$y_j = \begin{cases} x_j & \text{if } j \in X \\ R(\xi_j) & \text{if } j \in V \setminus (X \cup Y) \\ \xi_j & \text{if } j \in Y \end{cases}$$

output (state) of neuron $j \in V$

$R(\xi) = \text{ReLU}(\xi) = \max(0, \xi)$ activation function

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

excitation of $j \in V \setminus X$

$j_\leftarrow = \{i \in V \mid (i, j) \in E\}$ inputs to neuron $j \in V \setminus X$

$X = \{0, 1, \ldots, n\} \subseteq V$ input neurons including formal $0 \in X$ $(y_0 \equiv 1)$ for biases $w_{j0}$ for $j \in V \setminus X$

$\uparrow$

$(x_1, \ldots, x_n) \in \mathbb{R}^n$ external input to $\mathcal{N}$

w.l.o.g., excluding (max) pooling layers ( $\max(y_1, y_2) = R(y_1 - y_2) + y_2$ )

# Regression Error of Approximated DNNs

$\widetilde{\mathcal{N}}$ is an approximated DNN of $\mathcal{N}$, sharing the same input neurons ($\widetilde{X} = X$) and the same number of output neurons ($|\widetilde{Y}| = |Y|$) (tilde denotes parameters of $\widetilde{\mathcal{N}}$)

$\longrightarrow$ regression error under $L_1$ norm for an external input $(x_1, \ldots, x_n) \in \mathbb{R}^n$

$$E(x_1, \ldots, x_n) = \sum_{j \in Y} |y_j - \widetilde{y_j}| = \sum_{j \in Y} \left| \xi_j - \widetilde{\xi_j} \right|$$

**Weight Rounding**—an important example of approximated $\widetilde{\mathcal{N}}$:

the weights (including the biases) in $\mathcal{N}$ are rounded
(e.g. to a given number of binary digits in their floating-point representations)

$$\widetilde{w_{ji}} = w_{ji} + \delta_{ji} \quad \text{for } j \in V \setminus X \ \& \ i \in j_{\leftarrow}$$

where $\delta_{ji} \in \mathbb{R}$ is a real rounding error of weight $w_{ji}$

# Worst-Case Interval State-Bounds

$$a_j \le y_j \le b_j \quad \text{for } j \in V \setminus Y$$

- w.l.o.g., (bounded) external inputs $(x_1, \ldots, x_n) \in [0,1]^n$ (via linear mapping)

  $\rightarrow \; 0 = a_j \le y_j = x_j \le b_j = 1$ for $j \in X \setminus \{0\}$ $(a_0 = y_0 = b_0 = 1)$

- feedforward propagation of interval state-bounds:

$$a_j = R(a'_j)\,, \quad b_j = R(b'_j) \quad \text{for } j \in V \setminus (X \cup Y)\,, \quad \text{where}$$

$$a'_j = \sum_{\substack{i \in j_\leftarrow \\ w_{ji} < 0}} w_{ji} b_i + \sum_{\substack{i \in j_\leftarrow \\ w_{ji} > 0}} w_{ji} a_i\,, \quad b'_j = \sum_{\substack{i \in j_\leftarrow \\ w_{ji} < 0}} w_{ji} a_i + \sum_{\substack{i \in j_\leftarrow \\ w_{ji} > 0}} w_{ji} b_i$$

- w.l.o.g., $a_j = 0$ & $b_j > 0$ for $j \in V \setminus Y$ (otherwise, $j$ can be removed)

- these interval state-bounds are tight only for one neuron

**Theorem.** *It is NP-hard to find the tight bounds even for two layers.*

# Worst-Case Bounds on Weight-Rounding Error

**Main Idea:** for each $j \in V$, find worst-case bounds $\alpha_j \le 0 \le \beta_j$ caused by weight-rounding errors such that

$$y_j + \alpha_j \le \widetilde{y}_j \le y_j + \beta_j$$

holds for every $\widetilde{y}_i$ satisfying

$$y_i + \alpha_i \le \widetilde{y}_i \le y_i + \beta_i \quad \text{for } i \in j_{\leftarrow} \,, \quad \text{over all } y_i \in [a_i, b_i] \,:$$

- $\alpha_j = \beta_j = 0$ for $j \in X$ (input neurons with $j_{\leftarrow} = \emptyset$ unaffected by weight rounding)
- $\alpha_j = \min(0, \alpha'_j) \le 0 \,, \; \beta_j = \max(0, \beta'_j) \ge 0$ for $j \in V \setminus X$ ,

where

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

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

# Global Worst-Case Upper Bound on Weight Rounding Error

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

highly overestimated $\longrightarrow$ infeasible for practical use:

**Example:** fully connected 3-layer (784–2000–1000-10) NN $\mathcal{N}_1$ trained on MNIST with 32-bit weights, rounded to 16 bits in the approximated $\widetilde{\mathcal{N}_1}$

| 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] |

much larger in magnitude with each subsequent layer

in contrast, the actual error values are below $0.1$ for all test data points

**Corollary.** *It is NP-hard to find the maximum error of approximated DNNs (for any approximation, not only weight rounding).*

Idea of proof: by reduction from the maximum state problem (previous Theorem)

# Shortcut Weights

the excitation $\xi_j$ of any neuron $j \in V \setminus X$ is a continuous piecewise linear function of the external input (due to ReLU is piecewise linear)

$\rightarrow$ within a subset $\Xi \subseteq [0,1]^n$ of the input space, the excitation is a linear function of the input-neuron states:

$$\xi_j = \sum_{i \in X} W_{ji}\, y_i \quad \text{for } (y_1, \ldots, y_n) \in \Xi$$

where $W_{ji}$ are the coefficients of the linear function, referred to as the shortcut weights (bias) from input neurons $i \in X$ to neuron $j \in V \setminus X$

for input $(x_1, \ldots, x_n) \in [0,1]^n$, its neighborhood $\Xi_S$ is defined so that $\xi_j$ are linear for all $j \in V \setminus X$ with fixed shortcut weights, where

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

is the set of hidden neurons saturated at zero output, $y_j = R(\xi_j) = 0$

$\rightarrow$ $\Xi_S$ is a convex polytope—an intersection of finitely many half-spaces:

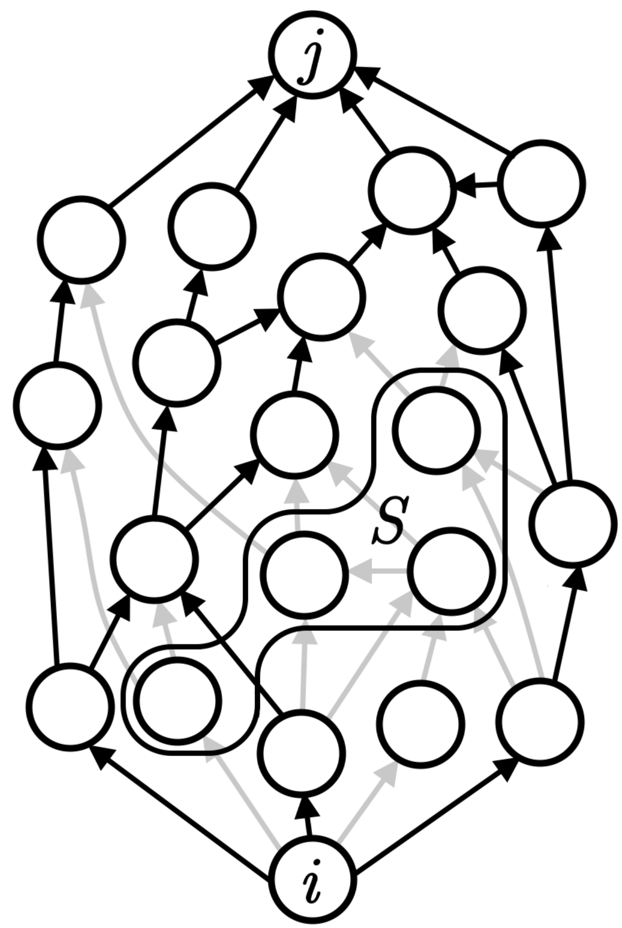
$$\xi_j = \sum_{i \in X} W_{ji}\, y_i \begin{cases} < 0 & \text{if } j \in S \\ \geq 0 & \text{if } j \notin S \end{cases} \quad \text{for } j \in V \setminus (X \cup Y)$$

$$0 \leq y_i \leq 1 \quad \text{for } i \in X$$

## Calculating Shortcut Weights

the shortcut weight $W_{ji}$ represents the cumulative influence from input neuron $i \in X$ to neuron $j \in V \setminus X$, corresponding to the product of weights along all connecting unsaturated paths in $\mathcal{N}$:

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




### Efficient Computation:

**1.** formal initialization for input neurons $j \in X$:

$$W_{ji} = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{otherwise} \end{cases} \quad \text{for all } i \in X$$

**2.** feedforward propagation for neurons $j \in V \setminus X$:

$$W_{ji} = \sum_{k \in j_{\leftarrow} \setminus S} w_{jk} W_{ki} \quad \text{for all } i \in X$$

# Estimating the Maximum Error of Approximated DNNs

- the worst-case maximum error does not take the probability distribution of the input space into account
- approximating the maximum or average error using data points from the training or test set $T$:

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

- refining the error estimate using the maximum over the convex polytope $\Xi_{S(x_1,\ldots,x_n)}$ surrounding the data point $(x_1,\ldots,x_n)\in T$:

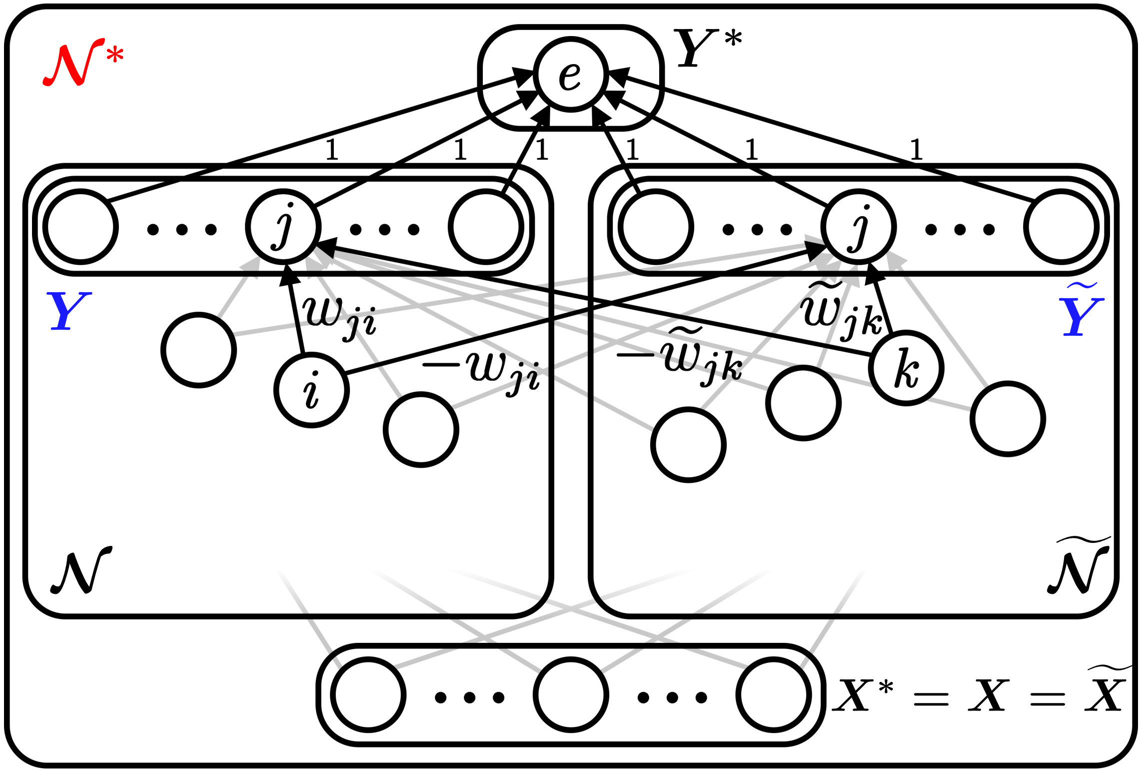
$$E_{\Xi_{S(T)}} = \max_{(x_1,\ldots,x_n)\in T} E_{\Xi_{S(x_1,\ldots,x_n)}}\,, \quad \overline{E_{\Xi_{S(T)}}} = \frac{1}{|T|}\sum_{(x_1,\ldots,x_n)\in T} E_{\Xi_{S(x_1,\ldots,x_n)}}$$

where

$$E_{\Xi_{S(x_1,\ldots,x_n)}} = \max_{(y_1,\ldots,y_n)\in \Xi_{S(x_1,\ldots,x_n)}} E(y_1,\ldots,y_n)$$

$\longrightarrow$ **AppMax method** for computing $E_{\Xi_{S(x_1,\ldots,x_n)}}$:

# Evaluating the Error of Approximated DNNs




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

$$\xi_j^* = \widetilde{\xi}_j - \xi_j \quad \text{for } j \in \widetilde{Y}$$
$$= \sum_{k \in j_\leftarrow} \widetilde{w_{jk}}\, y_k - \sum_{k \in j_\leftarrow} w_{jk} \widetilde{y_k}$$

$$y_e^* = \xi_e^* = \sum_{j \in Y} y_j^* + \sum_{j \in \widetilde{Y}} y_j^* = \sum_{j \in Y} R\left(\xi_j^*\right) + \sum_{j \in \widetilde{Y}} R\left(\xi_j^*\right)$$
$$= \sum_{j \in Y} \left( R\left(\xi_j - \widetilde{\xi}_j\right) + R\left(\widetilde{\xi}_j - \xi_j\right)\right) = \sum_{j \in Y} \left|\xi_j - \widetilde{\xi}_j\right| = E(x_1, \ldots, x_n)$$

# AppMax Method

**Input:** DNN $\mathcal{N}$, its approximation $\widetilde{\mathcal{N}}$, data point $(x_1,\ldots,x_n) \in T$

**Output:** $E_{\Xi_{S^*(x_1,\ldots,x_n)}} = \max_{(y_1,\ldots,y_n)\in\Xi_{S^*(x_1,\ldots,x_n)}} E(y_1,\ldots,y_n)$

**Algorithm:**

- construct $\mathcal{N}^*$ from $\mathcal{N}$ & $\widetilde{\mathcal{N}}$, computing the approximation error

$$y_e^* = E(x_1,\ldots,x_n) = \sum_{j\in Y} |y_j - \widetilde{y}_j|$$

- determine the saturated neurons $S^* = S^*(x_1,\ldots,x_n)$
- compute the shortcut weights $W_{ji}^*$ of $\mathcal{N}^*$ for all $j \in V^* \setminus X^*$ and $i \in X^*$
- solve the linear program (LP) to find the input-neuron states $y_1,\ldots,y_n$ that

$$\text{maximize} \quad y_e^* = \sum_{i\in X} W_{ei}^* y_i \quad \rightarrow \quad E_{\Xi_{S^*(x_1,\ldots,x_n)}}$$

$$\text{subject to} \quad \xi_j^* = \sum_{i\in X} W_{ji}^* y_i \begin{cases} \leq 0 & \text{if } j \in S^* \\ \geq 0 & \text{if } j \notin S^* \end{cases} \quad \text{for } j \in V^* \setminus (X^* \cup Y^*)$$

$$0 \leq y_i \leq 1 \quad \text{for } i \in X^*$$

## Experiments

- **Dataset:** MNIST handwritten digits (28x28 grayscale pixels) categorized into 10 classes (0–9)
- **Software Libraries:** PyTorch (deep learning), SciPy (linear programming)
- **Source Code:** publicly available at https://github.com/PetraVidnerova/RoundingErrorEstimation
- **DNNs:** trained on MNIST with 32-bit weights
  1. fully connected NN $\mathcal{N}_1$: 3 FC layers 784–2000–1000–10
  2. convolutional NN $\mathcal{N}_2$:
     - 2 convolutional layers with 32 and 64 3x3-kernels (stride 1, padding 1),
     - 1 max pooling layer with 64 2x2-kernels (stride 2)
     - 2 FC layers (1024–10)

     $\longrightarrow$ 8 FC layers 784–25088–50176–50176–25088–25088–12544–1024–10

robust accuracies of $\widetilde{\mathcal{N}}_1$ and $\widetilde{\mathcal{N}}_2$ on the test set for rounded weights:

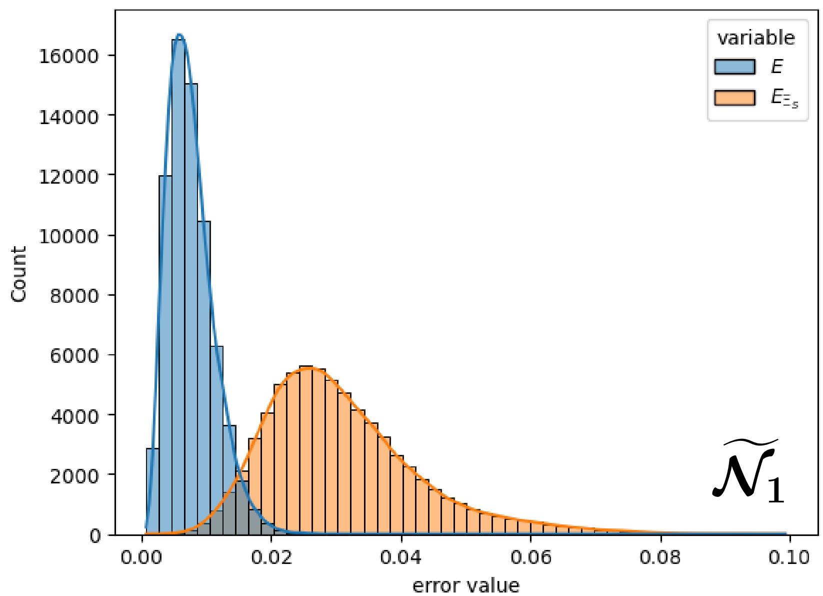
| weight bit-width | 32b | 16b | 12b | 8b | 6b | 4b |
|---|---|---|---|---|---|---|
| $\widetilde{\mathcal{N}}_1$ | 98.30 | 98.30 | 98.30 | 98.30 | 98.30 | 24.85 |
| $\widetilde{\mathcal{N}}_2$ | 99.25 | 99.25 | 99.25 | 99.25 | 99.25 | 99.14 |

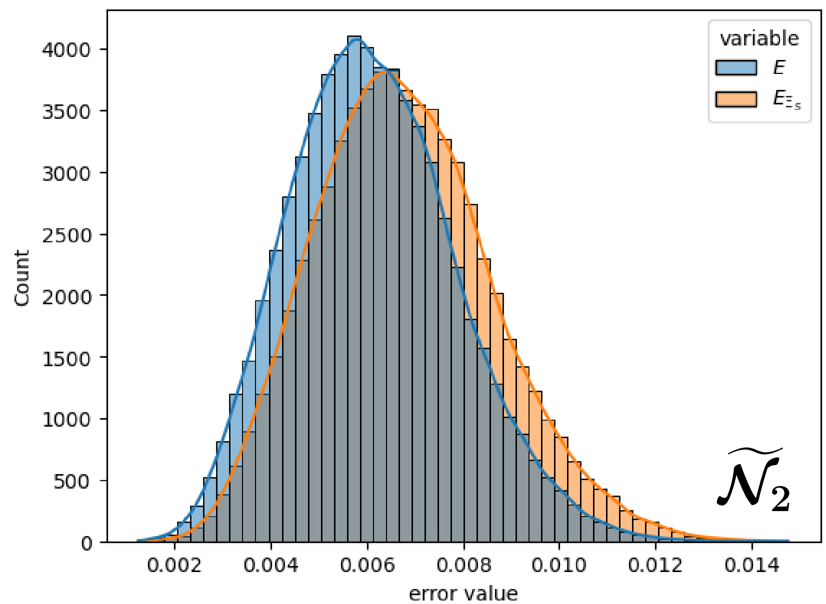
# Refining the Error Estimation Using the AppMax Method

- weights of approximated $\widetilde{\mathcal{N}}_1$ and $\widetilde{\mathcal{N}}_2$ rounded to **16 bits**
- test set $T$ contains all available 70,000 data points (i.e. 70,000 LPs)

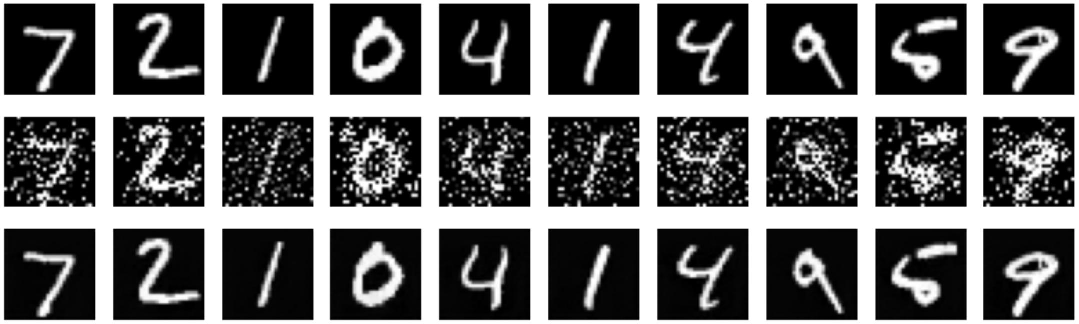
| | $E_T$ | $E_{\Xi_{S(T)}}$ | $\overline{E_T}$ | $\overline{E_{\Xi_{S(T)}}}$ |
|---|---|---|---|---|
| $\widetilde{\mathcal{N}}_1$ | 0.032854 | 0.099374 | 0.007629 | 0.030884 |
| $\widetilde{\mathcal{N}}_2$ | 0.013466 | 0.014763 | 0.006127 | 0.006777 |

Error Histograms: $E$ at data points in $T$ vs. $E_{\Xi_S}$ over convex polytopes $\Xi_S$ surrounding data points in $T$



# Worst-Case Points in Polytopes Identified by AppMax



data points $(x_1, \ldots, x_n) \in T$

$\widetilde{\mathcal{N}}_1$

$\widetilde{\mathcal{N}}_2$

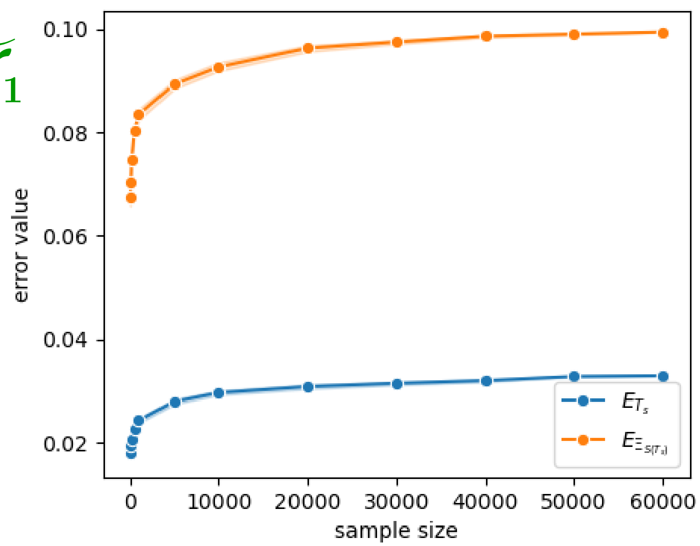
$$\underset{(y_1,\ldots,y_n)\in\Xi_{S(x_1,\ldots,x_n)}}{\arg\max} E(y_1, \ldots, y_n)$$

# Reducing the Sample Size for AppMax

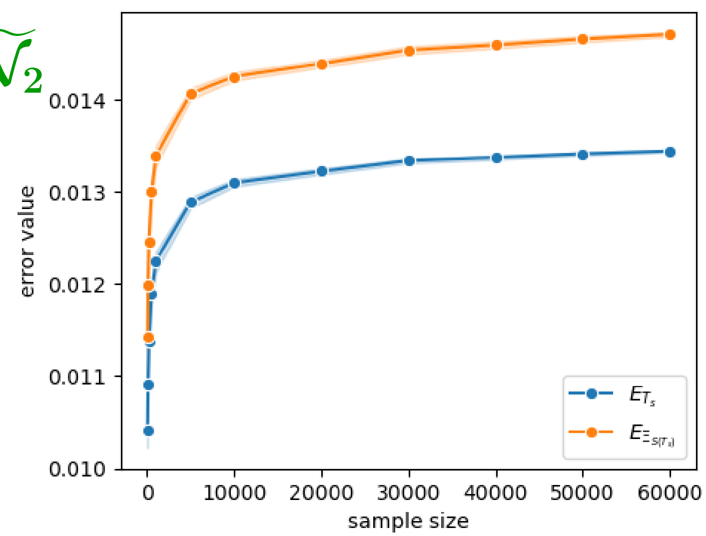
70,000 data points ($\sim$ LPs) required several days using dozens of parallel processors (e.g., $\widetilde{\mathcal{N}}_2$: 250 s per one data point on Intel® Xeon® E5-2620 v4 2.10 GHz processor)

error estimates $E_{T_s}$ and $E_{\Xi_{S(T_s)}}$ for random samples $T_s \subset T$ of increasing size (50–60,000), averaged over 100 trials:

$\widetilde{\mathcal{N}}_1$




$\widetilde{\mathcal{N}}_2$

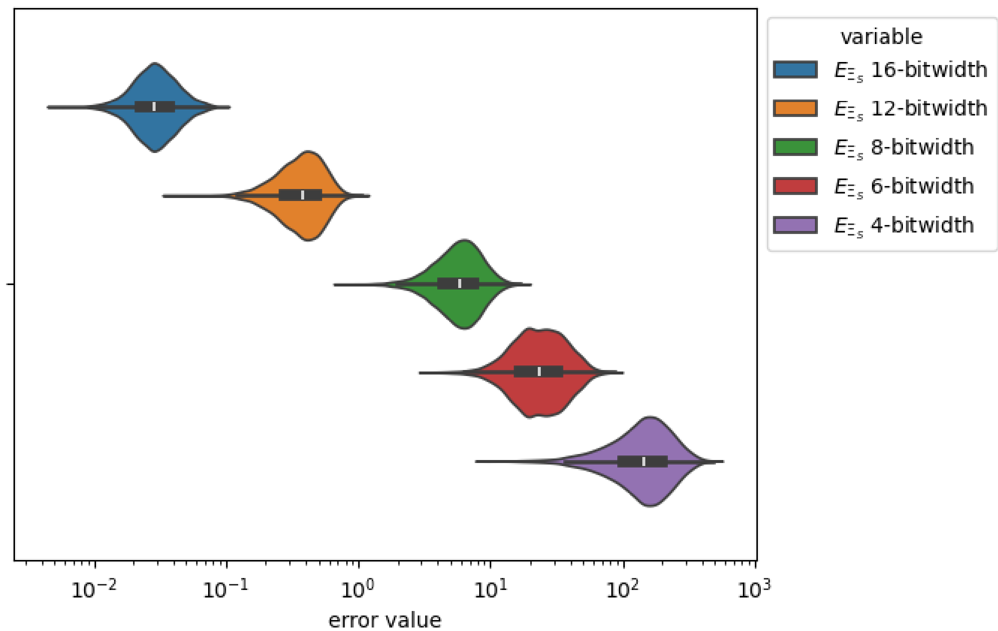


# Error Estimates for Decreasing Bit-Width of Weights

sample $T$ of 10,000 randomly chosen test points for $\widetilde{\mathcal{N}}_1$:

| weight bit-width | $E_T$ | $E_{\Xi_{S(T)}}$ | $\overline{E_T}$ | $\overline{E_{\Xi_{S(T)}}}$ |
|---|---|---|---|---|
| 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 |

violin plots of $E_{\Xi_S}$ (log scale) for different weight bit-widths:

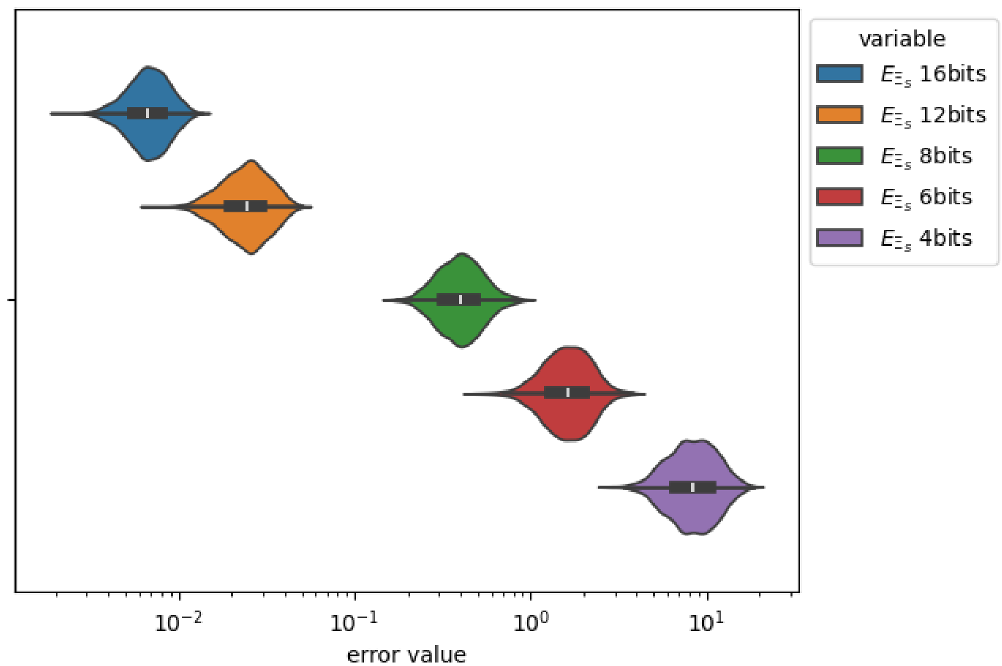


# Error Estimates for Decreasing Bit-Width of Weights

sample $T$ of 2,000 randomly chosen test points for $\widetilde{\mathcal{N}}_2$:

| weight bit-width | $E_T$ | $E_{\Xi_{S(T)}}$ | $\overline{E_T}$ | $\overline{E_{\Xi_{S(T)}}}$ |
|---|---|---|---|---|
| 16 bits | 0.012124 | 0.013333 | 0.006172 | 0.006853 |
| 12 bits | 0.044369 | 0.049109 | 0.022313 | 0.024935 |
| 8 bits | 0.821959 | 0.898328 | 0.368140 | 0.411297 |
| 6 bits | 3.522414 | 3.848394 | 1.486143 | 1.665951 |
| 4 bits | 16.409141 | 17.810625 | 7.662384 | 8.548645 |

violin plots of $E_{\Xi_S}$ (log scale) for different weight bit-widths:



## Summary

- theoretical analysis of the effect of weight-rounding on outputs of trained DNNs
- worst-case upper bound on weight-rounding error (overestimated in practice)
- computing regression error for approximated DNNs is NP-hard
- AppMax method: finds maximum error in convex polytopes around data points
- AppMax shows improved error guarantees (vs. test data only) on MNIST database for decreasing bit-width of weights
- AppMax enables comparison of approximation strategies, identifies optimal accuracy-performance tradeoffs, supports energy-efficient DNNs

## Future Research Directions

- AppMax for cross-entropy loss in classification DNNs with softmax via linear interpolation of $e^x$ (ICONIP 2025) vs. Karush-Kuhn-Tucker optimization ?
- approximate global error by estimating the probabilities of convex polytopes from their volumes measured by mean width
- broaden AppMax evaluation to other datasets (e.g., CIFAR-100, ImageNet)
- extend error analysis to modern architectures (e.g., ResNet, Transformers)
- identify DNN components that can be neglected (e.g., specific weights to be rounded) under explicitly bounded output error