PAPER Special Section on VLSI Design and CAD Algorithms

An Overflow/Underflow-Free Fixed-Point Bit-Width Optimization Method for OS-ELM Digital Circuit*

Mineto TSUKADA[†a)], ***Nonmember and*** **Hiroki MATSUTANI**[†b)], ***Member***

SUMMARY Currently there has been increasing demand for real-time training on resource-limited IoT devices such as smart sensors, which realizes standalone online adaptation for streaming data without data transfers to remote servers. OS-ELM (Online Sequential Extreme Learning Machine) has been one of promising neural-network-based online algorithms for on-chip learning because it can perform online training at low computational cost and is easy to implement as a digital circuit. Existing OS-ELM digital circuits employ fixed-point data format and the bit-widths are often manually tuned, however, this may cause overflow or underflow which can lead to unexpected behavior of the circuit. For on-chip learning systems, an overflow/underflow-free design has a great impact since online training is continuously performed and the intervals of intermediate variables will dynamically change as time goes by. In this paper, we propose an overflow/underflow-free bit-width optimization method for fixed-point digital circuits of OS-ELM. Experimental results show that our method realizes overflow/underflow-free OS-ELM digital circuits with 1.0x - 1.5x more area cost compared to the baseline simulation method where overflow or underflow can happen.

key words: *OS-ELM, bit-width optimization, fixed-point design*

1. Introduction

Currently there has been increasing demand for real-time training on resource-limited IoT devices (e.g. smart sensors and micro computers), which realizes standalone online adaptation for streaming data without transferring data to remote servers, and avoids additional power consumption for communication [1]. OS-ELM (Online Sequential Extreme Learning Machine) [2] has been one of promising neural-network-based online algorithms for on-chip learning because it can perform online training at low computational cost and is easy to implement as a digital circuit. Several papers have proposed design methodologies and implementations of OS-ELM digital circuits and shown that OS-ELM can be implemented in a small-size FPGA and still be able to perform online training in less than one millisecond [1], [3]–[5].

Existing OS-ELM digital circuits often employ fixed-point data format and the bit-widths are manually tuned according to the requirements (e.g. resource and timing constraints), however, this may cause overflow or underflow which can lead to unexpected behavior of the circuit. A lot of works have proposed bit-width optimization methods that analytically derive the lower and upper bounds of intermediate variables and automatically optimize the fixed-point data format, ensuring that overflow/underflow never happens [6]–[8]. For on-chip learning systems, an overflow/underflow-free design has a significant impact because online training is continuously performed and the intervals of intermediate variables will dynamically change as time goes by.

In this paper we propose an overflow/underflow-free bit-width optimization method for fixed-point OS-ELM digital circuits. This work makes the following contributions.

- We propose an interval analysis method for OS-ELM using affine arithmetic [9], one of the most widely-used interval arithmetic models. Affine arithmetic has been used in a lot of existing works for determining optimal integer bit-widths that never cause overflow and underflow.
- In affine arithmetic, division is defined only if the denominator does not include zero; otherwise the algorithm cannot be represented in affine arithmetic. OS-ELM's training algorithm contains one division and we analytically prove that the denominator does not include zero. Based on this proof, we also propose a mathematical trick to safely represent OS-ELM in affine arithmetic.
- Affine arithmetic can represent only fixed-length computation graphs and unbounded loops are not supported in affine arithmetic. However, OS-ELM's training algorithm is an iterative algorithm where current outputs are used as the next inputs endlessly. We propose an empirical solution for this problem based on simulation results, and verify its effectiveness in the evaluation section.
- We evaluate the performance of our interval analysis method, using an fixed-point IP core called OS-ELM Core to demonstrate the practicality of our method.

The rest of this paper is organized as follows; Section 2 gives a brief introduction of basic technologies behind this work. Our interval analysis method is proposed in Sect. 3. Section 4 briefly describes the design of OS-ELM Core. The proposed interval analysis method is evaluated in Sect. 5. Section 7 concludes this paper. Please refer to Table 4 and Table 5 for the notation rules and the description description of special variables that frequently appear in this paper.

Manuscript received March 15, 2021.
Manuscript revised July 13, 2021.
Manuscript publicized September 17, 2021.

[†]The authors are with Graduate School of Science and Technology, Keio University, Yokohama-shi, 223-8522 Japan.

*This work was partially supported by JST CREST Grant Number JPMJCR20F2, Japan.

a) E-mail: tsukada@arc.ics.keio.ac.jp
b) E-mail: matutani@arc.ics.keio.ac.jp

DOI: 10.1587/transfun.2021VLP0017

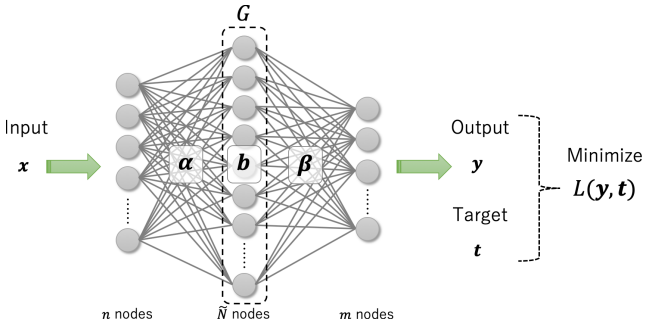


Fig. 1 Extreme learning machine. $n/\tilde{N}/m$ represents the number of input/hidden/output nodes. $\boldsymbol{\alpha} \in \mathbb{R}^{n\times\tilde{N}}$ is a weight matrix connecting the input and the hidden layers. $\boldsymbol{\beta} \in \mathbb{R}^{\tilde{N}\times m}$ is another weight matrix connecting the hidden and output layers. $\boldsymbol{b} \in \mathbb{R}^{1\times\tilde{N}}$ is a bias vector of the hidden layer, and G is an activation function applied to the hidden layer output. $\boldsymbol{\alpha}$ and $\boldsymbol{b}$ are non-trainable constant parameters initialized with random values.

2. Preliminaries

2.1 ELM

We first introduce ELM (Extreme Learning Machine) [10] prior to OS-ELM. ELM illustrated in Fig. 1 is a neural-network-based model that consists of an input layer, one hidden layer, and an output layer. If an n-dimensional input $\boldsymbol{X} \in \mathbb{R}^{k\times n}$ of batch size = k is given, the m-dimensional prediction output $\boldsymbol{Y} \in \mathbb{R}^{k\times m}$ can be computed in the following formula.

$$\boldsymbol{Y} = \mathrm{G}(\boldsymbol{X}\cdot\boldsymbol{\alpha} + \boldsymbol{b})\boldsymbol{\beta} \tag{1}$$

ELM uses a finite number of input-target pairs for training. Suppose an ELM model can approximate N input-target pairs $\{\boldsymbol{X} \in \mathbb{R}^{N\times n}, \boldsymbol{T} \in \mathbb{R}^{N\times m}\}$ with zero error, it implies there exists $\boldsymbol{\beta}$ that satisfies the following equation.

$$\mathrm{G}(\boldsymbol{X}\cdot\boldsymbol{\alpha} + \boldsymbol{b})\boldsymbol{\beta} = \boldsymbol{T} \tag{2}$$

Let $\boldsymbol{H} \equiv \mathrm{G}(\boldsymbol{X}\cdot\boldsymbol{\alpha}+\boldsymbol{b})$ then the optimal solution $\boldsymbol{\beta}^*$ is derived with the following formula.

$$\boldsymbol{\beta}^* = \boldsymbol{H}^\dagger\boldsymbol{T} \tag{3}$$

$\boldsymbol{H}^\dagger$ is the pseudo inverse of $\boldsymbol{H}$. The whole training process finishes by replacing $\boldsymbol{\beta}$ with $\boldsymbol{\beta}^*$. ELM takes one-shot optimization approach unlike backpropagation-based neural-networks (BP-NNs), which makes the whole training process faster. ELM is known to finish optimization process faster than BP-NNs [10].

2.2 OS-ELM

ELM is a batch learning algorithm; ELM needs to re-train with the whole training dataset, including training samples already learned in the past, in order to learn new training samples. OS-ELM [2] is an ELM variant that can perform online learning instead of batch learning. Suppose the ith training samples $\{\boldsymbol{X}_i \in \mathbb{R}^{k_i\times n}, \boldsymbol{T}_i \in \mathbb{R}^{k_i\times m}\}$ of batch size = k_i is given, OS-ELM computes the ith optimal solution $\boldsymbol{\beta}_i$ in the following formula.

$$\begin{aligned}\boldsymbol{P}_i &= \boldsymbol{P}_{i-1} - \boldsymbol{P}_{i-1}\boldsymbol{H}_i^T(\boldsymbol{I} + \boldsymbol{H}_i\boldsymbol{P}_{i-1}\boldsymbol{H}_i^T)^{-1}\boldsymbol{H}_i\boldsymbol{P}_{i-1}\\ \boldsymbol{\beta}_i &= \boldsymbol{\beta}_{i-1} + \boldsymbol{P}_i\boldsymbol{H}_i^T(\boldsymbol{T}_i - \boldsymbol{H}_i\boldsymbol{\beta}_{i-1}),\end{aligned} \tag{4}$$

where $\boldsymbol{H}_i \equiv G(\boldsymbol{X}_i\cdot\boldsymbol{\alpha} + \boldsymbol{b})$. $\boldsymbol{P}_0$ and $\boldsymbol{\beta}_0$ are computed as follows.

$$\begin{aligned}\boldsymbol{P}_0 &= (\boldsymbol{H}_0^T\boldsymbol{H}_0)^{-1}\\ \boldsymbol{\beta}_0 &= \boldsymbol{P}_0\boldsymbol{H}_0^T\boldsymbol{T}_0\end{aligned} \tag{5}$$

Note that OS-ELM and ELM produce the same solution as long as the training dataset is the same.

Specially, when $k_i = 1$ Eq. (4) can be simplified into

$$\begin{aligned}\boldsymbol{P}_i &= \boldsymbol{P}_{i-1} - \frac{\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T\boldsymbol{h}_i\boldsymbol{P}_{i-1}}{1 + \boldsymbol{h}_i\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T}\\ \boldsymbol{\beta}_i &= \boldsymbol{\beta}_{i-1} + \boldsymbol{P}_i\boldsymbol{h}_i^T(\boldsymbol{t}_i - \boldsymbol{h}_i\boldsymbol{\beta}_{i-1}),\end{aligned} \tag{6}$$

where $\boldsymbol{h}_i \equiv G(\boldsymbol{x}_i\cdot\boldsymbol{\alpha} + \boldsymbol{b})$. Note that $\boldsymbol{x}_i/\boldsymbol{t}_i/\boldsymbol{h}_i$ is a special case of $\boldsymbol{X}_i/\boldsymbol{T}_i/\boldsymbol{H}_i$ when $k_i = 1$. Equation (6) is more costless than Eq. (4) in terms of computational complexity since a costly matrix inverse $(\boldsymbol{I} + \boldsymbol{H}_i\boldsymbol{P}_{i-1}\boldsymbol{H}_i^T)^{-1}$ has been replaced with a simple reciprocal operation $\frac{1}{1+\boldsymbol{h}_i\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T}$ [1]. In this work we refer to Eq. (6) as "training algorithm" of OS-ELM. Equation (5) is referred to as "initialization algorithm".

The prediction algorithm is below.

$$\boldsymbol{y} = \mathrm{G}(\boldsymbol{x}\cdot\boldsymbol{\alpha} + \boldsymbol{b})\boldsymbol{\beta}, \tag{7}$$

where $\boldsymbol{y}$ is a special case of $\boldsymbol{Y}$ when the batch size is equal to 1. We refer to Eq. (7) as "prediction algorithm" of OS-ELM.

2.3 Interval Analysis

To realize an overflow/underflow-free fixed-point design you need to know the interval of each variable and allocate sufficient integer bits that never cause overflow and underflow. Existing interval analysis methods for fixed-point design are categorized into a (1) dynamic method or a (2) static method [11]. Dynamic methods [12]–[15] often take a simulation-based approach with tons of test inputs. It is known that dynamic methods often produce a better result close to the true interval compared to static methods, although they tend to take a long time due to exhaustive search and may encounter overflow or underflow if unseen inputs are found in runtime. Static methods [6]–[8], [16], [17], on the other hand, take a more analytical approach; they often involve solving equations and deriving upper and lower bounds of each variable without test inputs. Static methods produce a more conservative result (i.e. a wider interval) compared to dynamic methods, although the result is analytically guaranteed. In this work we employ a static method for interval analysis as the goal is to realize an overflow/underflow-free fixed-point OS-ELM digital circuit with analytical guarantee.

Interval arithmetic (IA) [18] is one of the oldest static interval analysis methods. In IA every variable is represented as an interval $[x_1, x_2]$ where x_1 and x_2 are the lower and upper bounds of the variable. Basic operations $\{+, -, \times\}$ are defined as follows;

$$\begin{aligned}
[x_1, x_2] + [y_1, y_2] &= [x_1 + y_1, x_2 + y_2] \\
[x_1, x_2] - [y_1, y_2] &= [x_1 - y_2, x_2 - y_1] \\
[x_1, x_2] \times [y_1, y_2] &= [\\
&\quad \min(x_1 y_1, x_1 y_2, x_2 y_1, x_2 y_2), \\
&\quad \max(x_1 y_1, x_1 y_2, x_2 y_1, x_2 y_2)]
\end{aligned} \tag{8}$$

IA guarantees intermediate intervals as long as input intervals are known. However, IA suffers from the *dependency problem*; for example, $x - x$ where $x \in [x_1, x_2]$ $(x_1 < x_2)$ should be 0 in ordinary algebra, although the result in IA is $[x_1 - x_2, x_2 - x_1]$, a wider interval than the true tightest range $[0, 0]$, which makes subsequent intervals get wider and wider. The cause of this problem is that IA ignores correlation of variables; $x - x$ is treated a self-subtraction in ordinary algebra but it is regarded as a subtraction between independent intervals in IA.

Affine arithmetic (AA) [9] is a refinement of IA proposed by Stolfi et al. AA keeps track of correlation of variables and is known to obtain tighter bounds close to the true range compared to IA. AA has been applied into a lot of fixed-point/floating-point bit-width optimization systems [6], [16], [19], [20] and still widely used in recent works [17], [21], [22]. We use AA throughout this work.

2.4 Affine Arithmetic

In AA the interval of variable x is represented in an *affine form* $\hat{x}$ given by;

$$\hat{x} = x_0 + x_1\epsilon_1 + x_2\epsilon_2 + \cdots + x_n\epsilon_n, \tag{9}$$

where $\epsilon_i \in [-1, 1]$. x_i is a coefficient and ϵ_i is an *uncertainty variable* which takes $[-1, 1]$; an affine form is a linear combination of uncertainty variables.

The interval of $\hat{x}$ can be computed as below.

$$\begin{aligned}
\text{interval}(\hat{x}) &= [\inf(\hat{x}), \sup(\hat{x})] \\
\inf(\hat{x}) &= x_0 - \sum_i |x_i| \\
\sup(\hat{x}) &= x_0 + \sum_i |x_i|
\end{aligned} \tag{10}$$

$\inf(\hat{x})$ computes the lower bound of $\hat{x}$ and $\sup(\hat{x})$ is the upper bound. Conversely a variable that ranges $[a, b]$ can be converted into an affine form $\hat{x} = x_0 + x_1\epsilon_1$ with

$$x_0 = \frac{b + a}{2}, \; x_1 = \frac{b - a}{2}. \tag{11}$$

Addition/subtraction between affine forms $\hat{x}$ and $\hat{y}$ is simply defined as $\hat{x} \pm \hat{y} = (x_0 \pm y_0) + \sum_i (x_i \pm y_i)\epsilon_i$. However, multiplication $\hat{x} * \hat{y}$ is a little bit complicated.

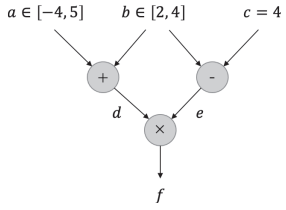


Fig. 2 simple tutorial of AA. In AA all input intervals $(a, b, c$ in this tutorial) must be known. Affine forms of a, b, c, d, e, f are computed as follows; $\hat{a} = 0.5 + 4.5\epsilon_a$, $\hat{b} = 3.0 + \epsilon_b$, $\hat{c} = 4.0$, $\hat{d} = 3.5 + 4.5\epsilon_a + \epsilon_b$, $\hat{e} = -1.0 + \epsilon_b$, $\hat{f} = -3.5 - 4.5\epsilon_a + 2.5\epsilon_b + 5.5\epsilon_f$. We can derive interval$(\hat{d}) = [-2.0, 9.0]$, interval$(\hat{e}) = [-2.0, 0.0]$, and interval$(\hat{f}) = [-16.0, 9.0]$ from Eq. (10).

$$\begin{aligned}
\hat{x} * \hat{y} &= x_0 y_0 + \sum_i (x_0 y_i + y_0 x_i)\epsilon_i + Q \\
Q &= \sum_i (x_i \epsilon_i) \sum_i (y_i \epsilon_i)
\end{aligned} \tag{12}$$

Note that Q is not an affine form (i.e. Q is not a linear combination of ϵ_i) and it needs approximation to become an affine form. A conservative approximation shown below is often taken [6], [16], [17].

$$Q \approx uv\epsilon_*, \; u = \sum_i |x_i|, \; v = \sum_i |y_i|, \tag{13}$$

where $\epsilon_* \in [-1, 1]$ is a new uncertainty variable. Note that $uv\epsilon_* \geq \sum_i (x_i\epsilon_i) \sum_i (y_i\epsilon_i)$. See Fig. 2 for a simple tutorial of AA.

Division $\hat{z} = \frac{\hat{x}}{\hat{y}}$ is often separated into $\hat{x} * \frac{1}{\hat{y}}$. There are mainly two approximation methods to compute $\frac{1}{\hat{y}}$: (1) the min-max approximation and (2) the chebyshev approximation. Here we show the definition of $\frac{1}{\hat{y}}$ with the min-max approximation.

$$\begin{aligned}
p &= \begin{cases} -\frac{1}{b^2} & (\text{if } b > a > 0) \\ -\frac{1}{a^2} & (\text{if } 0 > b > a) \end{cases} \\
q &= \frac{(a + b)^2}{2ab^2}, d = \frac{(a - b)^2}{2ab^2} \\
\frac{1}{\hat{y}} &= (p \cdot y_0 + q) + \sum_i p \cdot (y_i\epsilon_i) + d\epsilon_*,
\end{aligned} \tag{14}$$

where $a = \inf(\hat{y})$ and $b = \sup(\hat{y})$. Note that $\frac{1}{\hat{y}}$ is defined only if $b > a > 0$ or $0 > b > a$. The denominator $\hat{y}$ must not include zero.

2.5 Determination of Integer Bit-Width

Suppose we have an affine form $\hat{x}$, the minimum number of integer bits that never cause overflow and underflow is computed by;

$$\begin{aligned}
IB &= \lceil \log_2(\max(|\inf(\hat{x})|, |\sup(\hat{x})|) + 1) + \alpha, \\
\alpha &= \begin{cases} 1 & (\text{if signed}) \\ 0 & \text{else.} \end{cases}
\end{aligned} \tag{15}$$

IB represents the optimal integer bit-width.

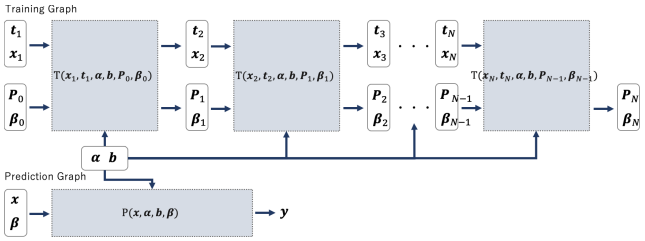


Fig. 3 Computation graphs for OS-ELM. N represents the total number of training steps.

Algorithm 1 $\mathrm{T}(\boldsymbol{x}_i, \boldsymbol{t}_i, \boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{P}_{i-1}, \boldsymbol{\beta}_{i-1}) \mapsto \{\boldsymbol{P}_i, \boldsymbol{\beta}_i\}$ $(1 \leq i \leq N)$.

Require: $\boldsymbol{x}_i, \boldsymbol{t}_i, \boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{P}_{i-1}, \boldsymbol{\beta}_{i-1}$
Ensure: $\boldsymbol{h}_i = \boldsymbol{x}_i \cdot \boldsymbol{\alpha} + \boldsymbol{b}$,
$\boldsymbol{P}_i = \boldsymbol{P}_{i-1} - \frac{\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T \boldsymbol{h}_i \boldsymbol{P}_{i-1}}{1+\boldsymbol{h}_i \boldsymbol{P}_{i-1} \boldsymbol{h}_i^T}$,
$\boldsymbol{\beta}_i = \boldsymbol{\beta}_{i-1} + \boldsymbol{P}_i \boldsymbol{h}_i^T (\boldsymbol{t}_i - \boldsymbol{h}_i \boldsymbol{\beta}_{i-1})$
$\boldsymbol{e}_i \leftarrow \boldsymbol{x}_i \cdot \boldsymbol{\alpha}$
$\boldsymbol{h}_i \leftarrow \boldsymbol{e}_i + \boldsymbol{b}$
$\boldsymbol{\gamma}_i^{(1)} \leftarrow \boldsymbol{P}_{i-1} \cdot \boldsymbol{h}_i^T$
$\boldsymbol{\gamma}_i^{(2)} \leftarrow \boldsymbol{h}_i \cdot \boldsymbol{P}_{i-1}$
$\boldsymbol{\gamma}_i^{(3)} \leftarrow \boldsymbol{\gamma}_i^{(1)} \cdot \boldsymbol{\gamma}_i^{(3)}$
$\gamma_i^{(4)} \leftarrow \boldsymbol{\gamma}_i^{(2)} \cdot \boldsymbol{h}_i^T$
$\gamma_i^{(5)} \leftarrow \gamma_i^{(4)} + 1$
$\boldsymbol{\gamma}_i^{(6)} \leftarrow \boldsymbol{\gamma}_i^{(3)} / \gamma_i^{(5)}$
$\boldsymbol{P}_i \leftarrow \boldsymbol{P}_i - \boldsymbol{\gamma}_i^{(6)}$
$\boldsymbol{\gamma}_i^{(7)} \leftarrow \boldsymbol{P}_i \cdot \boldsymbol{h}_i^T$
$\boldsymbol{\gamma}_i^{(8)} \leftarrow \boldsymbol{h}_i \cdot \boldsymbol{\beta}_{i-1}$
$\boldsymbol{\gamma}_i^{(9)} \leftarrow \boldsymbol{t}_i - \boldsymbol{\gamma}_i^{(8)}$
$\boldsymbol{\gamma}_i^{(10)} \leftarrow \boldsymbol{\gamma}_i^{(7)} \cdot \boldsymbol{\gamma}_i^{(9)}$
$\boldsymbol{\beta}_i \leftarrow \boldsymbol{\beta}_{i-1} + \boldsymbol{\gamma}_i^{(10)}$
return $\{\boldsymbol{P}_i, \boldsymbol{\beta}_i\}$

3. AA-Based Interval Analysis for OS-ELM

In this section we propose the AA-based interval analysis method for OS-ELM. The process is two-fold: ① Build the computation graph equivalent to OS-ELM. ② Compute the affine form and interval for every variable existing in OS-ELM, using Eq. (10). Figure 3 shows computation graphs for OS-ELM. "Training graph" corresponds to the training algorithm (Eq. (6)), and "prediction graph" corresponds to the prediction algorithm (Eq. (7)).

$\mathrm{T}(\boldsymbol{x}_i, \boldsymbol{t}_i, \boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{P}_{i-1}, \boldsymbol{\beta}_{i-1}) \mapsto \{\boldsymbol{P}_i, \boldsymbol{\beta}_i\}$ defined in Algorithm 1 represents a sub-graph that computes a single iteration of the OS-ELM training algorithm. Training graph concatenates N sub-graphs, where N is the total number of training steps. Training graph takes $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \boldsymbol{t}_1, \ldots, \boldsymbol{t}_N, \boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{P}_0, \boldsymbol{\beta}_0\}$ as input and outputs $\{\boldsymbol{P}_N, \boldsymbol{\beta}_N\}$. $\mathrm{P}(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{\beta}) \mapsto \boldsymbol{y}$ defined in Algorithm 2 represents prediction graph. Prediction graph takes $\{\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{\beta}\}$ as input and outputs $\boldsymbol{y}$.

The goal is to obtain the intervals of $\{\boldsymbol{\gamma}_i^{(1)}, \ldots, \boldsymbol{\gamma}_i^{(10)}, \boldsymbol{P}_i, \boldsymbol{\beta}_i, \boldsymbol{e}_i, \boldsymbol{h}_i\}$ $(1 \leq i \leq N)$ for training graph and $\{\boldsymbol{e}, \boldsymbol{h}, \boldsymbol{y}\}$ for prediction graph, through AA. In this paper, the interval of a matrix $\boldsymbol{A} \in \mathbb{R}^{u \times v}$ is computed as follows.

Algorithm 2 $\mathrm{P}(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{\beta}) \mapsto \boldsymbol{y}$

Require: $\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{\beta}$
Ensure: $\boldsymbol{y} = (\boldsymbol{x} \cdot \boldsymbol{\alpha} + \boldsymbol{b})\boldsymbol{\beta}$
$\boldsymbol{e} \leftarrow \boldsymbol{x} \cdot \boldsymbol{\alpha}$
$\boldsymbol{h} \leftarrow \boldsymbol{e} + \boldsymbol{b}$
$\boldsymbol{y} \leftarrow \boldsymbol{h} \cdot \boldsymbol{\beta}$
return $\boldsymbol{y}$

$$
\begin{aligned}
\mathrm{interval}(\hat{\boldsymbol{A}}) &= [\inf(\hat{\boldsymbol{A}}), \sup(\hat{\boldsymbol{A}})] \\
\inf(\hat{\boldsymbol{A}}) &= \min(\inf(\hat{A}_{[0,0]}), \ldots, \inf(\hat{A}_{[u-1,v-1]})) \\
\sup(\hat{\boldsymbol{A}}) &= \max(\sup(\hat{A}_{[0,0]}), \ldots, \sup(\hat{A}_{[u-1,v-1]})),
\end{aligned} \tag{16}
$$

where $\hat{\boldsymbol{A}}$ is the affine form of $\boldsymbol{A}$, and $\hat{A}_{[i,j]}$ is the ij element of $\hat{\boldsymbol{A}}$.

3.1 Constraints

Remember that all input intervals must be known in AA; in other words the intervals of $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \boldsymbol{t}_1, \ldots, \boldsymbol{t}_N, \boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{P}_0, \boldsymbol{\beta}_0\}$ for training graph and $\{\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{\beta}\}$ for prediction graph must be given. In this work we assume that the intervals of $\{\boldsymbol{x}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \boldsymbol{t}_1, \ldots, \boldsymbol{t}_N\}$ are $[0, 1]$, and those of $\{\boldsymbol{\alpha}, \boldsymbol{b}\}$ are $[-1, 1]$. $\{\boldsymbol{P}_0, \boldsymbol{\beta}_0\}$ is computed by Eq. (5). The interval of $\boldsymbol{\beta}$ (an input of prediction graph) is computed in the way described in Sect. 3.3.

3.2 Interval Analysis for Training Graph

The goal of training graph is to find the intervals of $\{\boldsymbol{\gamma}_i^{(1)}, \ldots, \boldsymbol{\gamma}_i^{(10)}, \boldsymbol{P}_i, \boldsymbol{\beta}_i, \boldsymbol{e}_i, \boldsymbol{h}_i\}$ for $1 \leq i \leq N$, however, we have to deal with a critical problem; OS-ELM is an online learning algorithm and the total number of training steps N is unknown as training may occur in runtime (i.e. N can increase in runtime). if N is unknown, the training graph grows endlessly and interval analysis becomes infeasible. We need to determine a "reasonable" value of N for training graph.

3.2.1 Determination of N

To determine N, we conducted an experiment to analyze the intervals of $\{\boldsymbol{\gamma}_i^{(1)}, \ldots, \boldsymbol{\gamma}_i^{(10)}, \boldsymbol{P}_i, \boldsymbol{\beta}_i, \boldsymbol{e}_i, \boldsymbol{h}_i\}$ for $1 \leq i \leq N$. The procedure is as follows: ① Implement OS-ELM's initialization and training algorithms in double-precision format. ② Compute initialization algorithm using initial training samples of Digits [23] dataset (see Table 1 for details). $\{\boldsymbol{P}_0, \boldsymbol{\beta}_0\}$ is obtained. ③ Compute training algorithm *by one step* using online training samples. $\{\boldsymbol{P}_k, \boldsymbol{\beta}_k\}$ is obtained if $i = k$. ④ Generate 1,000 random training samples $\{\boldsymbol{x}, \boldsymbol{t}\}$ with uniform distribution of [0, 1]. Feed all the random samples into training algorithm of step = k and measure the maximum and minimum values for each of $\{\boldsymbol{\gamma}_k^{(1)}, \ldots, \boldsymbol{\gamma}_k^{(10)}, \boldsymbol{P}_k, \boldsymbol{\beta}_k, \boldsymbol{e}_k, \boldsymbol{h}_k\}$.

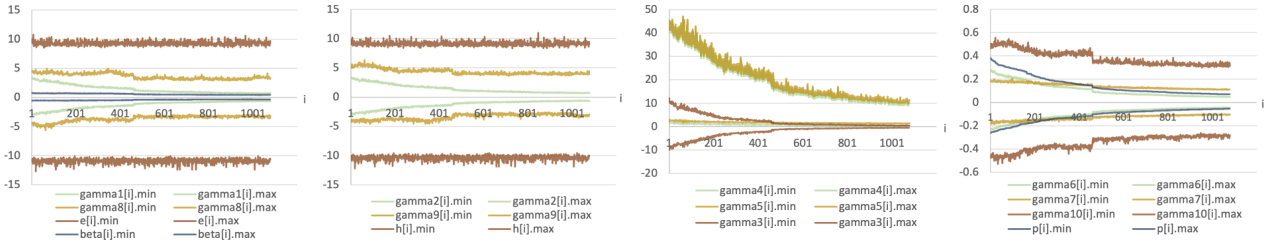


Fig. 4 Observed intervals of $\{\boldsymbol{\gamma}_i^{(1)}, \ldots, \boldsymbol{\gamma}_i^{(10)}, \boldsymbol{P}_i, \boldsymbol{\beta}_i, \boldsymbol{e}_i, \boldsymbol{h}_i\}$ $(1 \le i \le N = 1,079)$ on Digits dataset. The x-axis represents the training step i, and the y-axis plots the observed intervals (the maximum and minimum values) of each variable at training step i.

Table 1 Classification datasets used in Sect. 5. "Initial training samples' refers to the training samples used for computing $\{\boldsymbol{\beta}_0, \boldsymbol{P}_0\}$. "Online training samples" are the training samples for computing $\{\boldsymbol{\beta}_i, \boldsymbol{P}_i\}$ $(i \ge 1)$. "Test samples" are used to evaluate test accuracy and determine the number of hidden nodes. "Features" is the number of dimension of input $\boldsymbol{x}$. "Classes" corresponds to the number of output classes (= the number of dimension of output $\boldsymbol{y}$ and target $\boldsymbol{t}$). "Model size" column shows the model size $\{n, \tilde{N}, m\}$ for each dataset, where n, $\tilde{N}$, or m represents the number of input, hidden, or output nodes.

Name	Initial training samples	Online training samples	Test samples	Features	Classes	Model size
Digits [23]	358	1,079	360	64	10	{64, 48, 10}
Iris [24]	30	90	30	4	3	{4, 5, 3}
Letter [25]	4,000	12,000	4,000	16	26	{16, 32, 26}
Credit [26]	6,000	18,000	6,000	23	2	{23, 16, 2}
Drive [27]	11,701	35,106	11,702	48	11	{48, 64, 11}

⑤ Iterate 3-4 until all online training samples run out.

Figure 4 shows the result. We observed that all the intervals gradually converged or kept constant as i proceeds. Similar outcomes were observed on other datasets too (see Sect. 5.3 for the entire result on multiple datasets). From these outcomes, we make a hypothesis that $\boldsymbol{A}_i \in \{\boldsymbol{\gamma}_i^{(1)}, \ldots, \boldsymbol{\gamma}_i^{(10)}, \boldsymbol{P}_i, \boldsymbol{\beta}_i, \boldsymbol{e}_i, \boldsymbol{h}_i\}$ roughly satisfies $[\min(\boldsymbol{A}_1), \max(\boldsymbol{A}_1)] \supseteq [\min(\boldsymbol{A}_i), \max(\boldsymbol{A}_i)]$ for $2 \le i$, in other words, the interval of $\boldsymbol{A}_1$ can be used as those of $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_N$. This hypothesis is verified in Sect. 5.3, using multiple datasets.

Based on the hypothesis we set $N = 1$ in training graph. The interval analysis method for training graph is summarized as follows.

1. Build training graph $\mathrm{T}(\boldsymbol{x}_0, \boldsymbol{t}_0, \boldsymbol{\alpha}, \boldsymbol{b}, \boldsymbol{P}_0, \boldsymbol{\beta}_0) \mapsto \{\boldsymbol{P}_1, \boldsymbol{\beta}_1\}$.
2. Compute $\{\hat{\boldsymbol{\gamma}}_1^{(1)}, \ldots, \hat{\boldsymbol{\gamma}}_1^{(10)}, \hat{\boldsymbol{P}}_1, \hat{\boldsymbol{\beta}}_1, \hat{\boldsymbol{e}}_1, \hat{\boldsymbol{h}}_1\}$ using AA. The intervals are used as those of $\{\boldsymbol{\gamma}_i^{(1)}, \ldots, \boldsymbol{\gamma}_i^{(10)}, \boldsymbol{P}_i, \boldsymbol{\beta}_i, \boldsymbol{e}_i, \boldsymbol{h}_i\}$ $(i \ge 1)$.

3.2.2 Division

OS-ELM's training algorithm has a division $\frac{\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T\boldsymbol{h}_i\boldsymbol{P}_{i-1}}{1+\boldsymbol{h}_i\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T}$. As mentioned in Sect. 2.4, the denominator $\gamma_i^{(5)} = 1 + \boldsymbol{h}_i\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T$ must not take zero. In the rest of this section $0 \notin \gamma_i^{(5)}$ is proven for $i \ge 1$.

Theorem 1. *$\boldsymbol{P}_{i-1}$ is positive-definite for $i \ge 1$.*

Proof. We first prove that $\boldsymbol{P}_0$ is positive-definite.

- $\boldsymbol{P}_0^{-1}$ is positive-semidefinite due to $\boldsymbol{u}\boldsymbol{P}_0^{-1}\boldsymbol{u}^T = \boldsymbol{u}\boldsymbol{H}_0^T\boldsymbol{H}_0\boldsymbol{u}^T = (\boldsymbol{u}\boldsymbol{H}_0^T)\cdot(\boldsymbol{u}\boldsymbol{H}_0^T)^T \ge 0$, where $\boldsymbol{u} \in \mathbb{R}^{1\times\tilde{N}}$ represents an arbitrary vector.
- $\boldsymbol{P}_0^{-1} = \boldsymbol{H}_0^T\boldsymbol{H}_0$ is positive-definite since $\boldsymbol{H}_0^T\boldsymbol{H}_0$ is assumed to be a regular matrix in OS-ELM.
- $\boldsymbol{P}_0$ is positive-definite since the inverse of a positive-definite matrix is positive-definite.

Next, we prove that $\boldsymbol{P}_1$ is positive-definite. Equation (17) is derived by applying the sherman-morrison formula[†] to Eq. (6).

$$\boldsymbol{P}_i = (\boldsymbol{P}_{i-1}^{-1} + \boldsymbol{h}_i^T\boldsymbol{h}_i)^{-1} \tag{17}$$

- $\boldsymbol{P}_1 = (\boldsymbol{P}_0^{-1} + \boldsymbol{h}_1^T\boldsymbol{h}_1)^{-1}$ holds by substituting $i = 1$.
- $\boldsymbol{h}_1^T\boldsymbol{h}_1$ is positive-semidefinite due to $\boldsymbol{u}\boldsymbol{h}_1^T\boldsymbol{h}_1\boldsymbol{u}^T = (\boldsymbol{u}\boldsymbol{h}_1^T)\cdot(\boldsymbol{u}\boldsymbol{h}_1^T)^T \ge 0$.
- $\boldsymbol{P}_1^{-1} = (\boldsymbol{P}_0^{-1} + \boldsymbol{h}_1^T\boldsymbol{h}_1)$ is positive-definite since it is the sum of a positive-definite matrix $\boldsymbol{P}_0^{-1}$ and a positive-semidefinite matrix $\boldsymbol{h}_1^T\boldsymbol{h}_1$.
- $\boldsymbol{P}_1$ is positive-definite since it is the inverse of a positive-definite matrix $\boldsymbol{P}_1^{-1}$.

By repeating the above logic, $\boldsymbol{P}_0, \ldots, \boldsymbol{P}_{i-1}$ $(i \ge 1)$ are all positive-definite. □

Theorem 2. *$\boldsymbol{h}_i\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T > 0$ for $i \ge 1$.*

Proof. An $n \times n$ positive-definite matrix $\boldsymbol{V} \in \mathbb{R}^{n\times n}$ satisfies the following inequality.

$$\boldsymbol{u}\boldsymbol{V}\boldsymbol{u}^T > 0, \tag{18}$$

where $\boldsymbol{u} \in \mathbb{R}^{1\times n}$ represents an arbitrary vector. By applying

[†] $(\boldsymbol{V} + \boldsymbol{u}^T\boldsymbol{w})^{-1} = \boldsymbol{V}^{-1} - \frac{\boldsymbol{V}^{-1}\boldsymbol{u}^T\boldsymbol{w}\boldsymbol{V}^{-1}}{1+\boldsymbol{w}\boldsymbol{V}^{-1}\boldsymbol{u}^T}$ $(\boldsymbol{V} \in \mathbb{R}^{k\times k}, \boldsymbol{u} \in \mathbb{R}^{1\times k}, \boldsymbol{w} \in \mathbb{R}^{1\times k}, k \in \mathbb{N})$.

this to $\boldsymbol{h}_i\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T$, $\boldsymbol{h}_i\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T > 0$ holds for $i \geq 1$, which guarantees $0 \notin 1 + \boldsymbol{h}_i\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T \Leftrightarrow 0 \notin \gamma_i^{(5)}$ for $i \geq 1$. □

Note that interval($\hat{\gamma}_i^{(5)}$) can include zero because interval($\hat{\gamma}_i^{(5)}$) can be wider than the true interval of $\gamma_i^{(5)}$. To tackle this problem we propose to compute $\min(1, \inf(\hat{\gamma}_i^{(5)}))$ for the lower bound of $\hat{\gamma}_i^{(5)}$ instead of $\inf(\hat{\gamma}_i^{(5)})$. This trick prevents $\hat{\gamma}_i^{(5)}$ from including zero and at the same time makes the interval close to the true interval. Thanks to this trick OS-ELM's training algorithm can be safely represented in AA.

3.3 Interval Analysis for Prediction Graph

Prediction graph takes $\{\boldsymbol{x}, \boldsymbol{\beta}\}$ as input. The interval of $\boldsymbol{\beta}$ should be that of $\boldsymbol{\beta}_i$ over $0 \leq i \leq N$, more specifically, $0 \leq i \leq 1$ ($N = 1$). We propose to compute $\min(\inf(\hat{\beta}_{0[u,v]}), \inf(\hat{\beta}_{1[u,v]}))$ for the lower bound of $\beta_{[u,v]}$ and $\max(\sup(\hat{\beta}_{0[u,v]}), \sup(\hat{\beta}_{1[u,v]}))$ as the upper bound, where $\beta_{[u,v]}$ represents the uv element of $\boldsymbol{\beta}$.

4. OS-ELM Core

We developed OS-ELM Core, a fixed-point IP core that implements OS-ELM algorithms, to verify the proposed interval analysis method. All integer bit-widths of OS-ELM Core are parametrized, and the result of proposed interval analysis method is used as the arguments. The PYNQ-Z1 FPGA board [28] (280 BRAM blocks, 220 DSP slices, 106,400 flip-flops, and 53,200 LUT instances) is employed as the evaluation platform.

Figure 5 shows the block diagram of OS-ELM Core. OS-ELM Core employs axi-stream protocol for input/output interface with 64-bit data width. Training module executes OS-ELM's training algorithm then updates $\boldsymbol{P}$ and $\boldsymbol{\beta}$ managed in parameter buffer. Prediction module reads an input $\boldsymbol{x}$ from input buffer and executes prediction algorithm. The output of prediction module $\boldsymbol{y}$ is buffered in output buffer. Both training and prediction modules use one adder and one multiplier in a matrix product operation, and one arithmetic unit (i.e. adder, multiplier, or divisor) in a element-wise operation, regardless of the size of matrix, to make hardware resource cost as small as possible. All the arrays existing in OS-ELM Core are implemented with BRAM blocks (18 kb/block), and all the fixed-point arithmetic units (i.e. adder, multiplier, and divisor) are with DSP slices.

5. Evaluation

In this section we evaluate the proposed interval analysis method. All the experiments here were executed on a server machine (Ubuntu 20.04, Intel Xeon E5-1650 3.60 GHz, DRAM 64GB, SSD 500 GB). Table 1 lists the classification datasets used for evaluation of our method. For all the datasets, the intervals of input $\boldsymbol{x}$ and target $\boldsymbol{t}$ are normalized

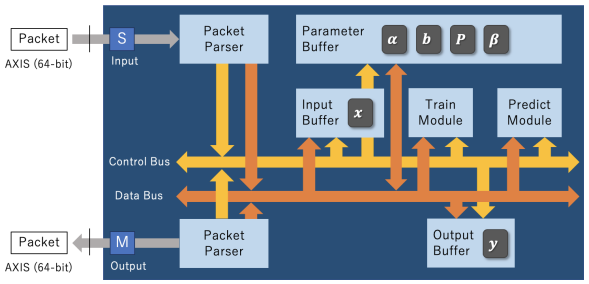


Fig. 5 Block diagram of OS-ELM Core.

into $[0, 1]$. Parameters $\boldsymbol{b}$ and $\boldsymbol{\alpha}$ are randomly generated with the uniform distribution of $[-1, 1]$. The model size for each dataset is shown in "Model Size" column. The number of hidden nodes is set to the number that performed the best test accuracy in a given search space; search spaces for Digits, Iris, Letter, Credit, and Drive are {32, 48, 64, 96, 128}, {3, 4, 5, 6, 7}, {8, 16, 32, 64, 128}, {4, 8, 16, 32, 64}, and {32, 64, 96, 128} respectively.

5.1 Optimization Result

In this section we first show the result of the proposed interval analysis method for each dataset, comparing with an ordinary simulation-based interval analysis method. Here is a brief introduction of the simulation method: ① Implement OS-ELM's initialization, prediction, and training algorithms in double-precision format. ② Execute initialization algorithm using initial training samples. $\{\boldsymbol{P}_0, \boldsymbol{\beta}_0\}$ is obtained. ③ Execute training algorithm by one step using online training samples. $\{\boldsymbol{P}_k, \boldsymbol{\beta}_k\}$ is obtained if $i = k$. ④ Generate 1,000 random training samples $\{\boldsymbol{x}, \boldsymbol{t}\}$ with uniform distribution of $[0, 1]$. ⑤ Feed all the random samples into training algorithm of step $= k$ and measure the values of $\{\boldsymbol{\gamma}_k^{(1)}, \ldots, \boldsymbol{\gamma}_k^{(10)}, \boldsymbol{P}_k, \boldsymbol{\beta}_k, \boldsymbol{e}_k, \boldsymbol{h}_k\}$. ⑥ Feed all the random samples into prediction algorithm and measure the values of $\boldsymbol{y}$. ⑦ Repeat 3-6 until all online training samples run out.

Table 2 shows the intervals obtained from the simulation method (sim) and those from the proposed method (ours). All the intervals obtained from our method cover the corresponding simulated interval. Note that the simulated interval of $\boldsymbol{\gamma}_i^{(5)} = 1 + \boldsymbol{h}_i\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T$ satisfies $\boldsymbol{\gamma}_i^{(5)} > 1$, which is consistent with the theorem proven in Sect. 3.2.2.

5.2 Rate of Overflow/Underflows

This section compares the simulation method introduced in Sect. 5.1 and the proposed method in terms of the rate of overflow/underflows, using OS-ELM Core. The experimental procedure is as follows: ① Execute the simulation method and convert the result into integer bit-widths using Eq. (15) (an extra bit was added to each bit-width to reduce overflow/underflows). ② Execute the proposed method and convert the result into bit-widths. ③ Synthesize two OS-ELM Cores using the bit-widths obtained from 1 and 2. ④ Execute training by one step in both OS-ELM Cores using

Table 2 Intervals obtained from simulation (sim) and the proposed interval analysis method (ours) for each dataset.

	$\gamma_i^{(1)}$	$\gamma_i^{(2)}$	$\gamma_i^{(3)}$	$\gamma_i^{(4)}$	$\gamma_i^{(5)}$
Digits (sim)	$[-0.642, 0.694]$	$[-0.642, 0.694]$	$[-0.446, 0.482]$	$[0.371, 9.75]$	$[1.37, 10.7]$
Digits (ours)	$[-9.92e^3, 9.91e^3]$	$[-9.26, 9.69]$	$[-24.5, 27.8]$	$[0.0, 1.46e^3]$	$[1.0, 1.46e^3]$
Iris (sim)	$[-5.94, 5.85]$	$[-5.94, 5.85]$	$[-4.89, 35.3]$	$[9.27e^{-3}, 3.24]$	$[1.01, 4.24]$
Iris (ours)	$[-1.55e^3, 1.55e^3]$	$[-63.5, 19.1]$	$[-388, 388]$	$[0.0, 48.0]$	$[1.0, 41.7]$
Letter (sim)	$[-6.72e^{-3}, 7.54e^{-3}]$	$[-6.72e^{-3}, 7.54e^{-3}]$	$[-5.06e^{-5}, 5.68e^{-5}]$	$[2.79e^{-3}, 0.0397]$	$[1.0, 1.04]$
Letter (ours)	$[-0.301, 0.307]$	$[-0.0593, 0.0785]$	$[-2.42e^{-3}, 2.44e^{-3}]$	$[0.0, 3.49]$	$[1.0, 4.49]$
Credit (sim)	$[-0.115, 0.116]$	$[-0.115, 0.116]$	$[-8.36e^{-3}, 0.0135]$	$[5.89e^{-3}, 0.253]$	$[1.01, 1.25]$
Credit (ours)	$[-32.9, 32.9]$	$[-2.22, 3.25]$	$[-0.589, 0.589]$	$[0.0, 32.4]$	$[1.0, 33.4]$
Drive (sim)	$[-6.97e^5, 6.92e^5]$	$[-6.98e^5, 6.92e^5]$	$[-3.71e^{11}, 4.87e^{11}]$	$[5.26e^4, 4.72e^6]$	$[5.26e^4, 4.72e^6]$
Drive (ours)	$[-6.56e^{15}, 6.56e^{15}]$	$[-1.33e^7, 1.56e^7]$	$[-1.4e^{13}, 1.4e^{13}]$	$[0.0, 1.55e^9]$	$[1.0, 1.55e^9]$
	$\gamma_i^{(6)}$	$\gamma_i^{(7)}$	$\gamma_i^{(8)}$	$\gamma_i^{(9)}$	$\gamma_i^{(10)}$
Digits (sim)	$[-0.0447, 0.0472]$	$[-0.102, 0.109]$	$[-3.25, 3.29]$	$[-3.0, 3.94]$	$[-0.291, 0.306]$
Digits (ours)	$[-25.8, 27.8]$	$[-9.92e^3, 9.91e^3]$	$[-12.1, 15.4]$	$[-8.38, 9.0]$	$[-8.93e^4, 8.93e^4]$
Iris (sim)	$[-1.32, 8.32]$	$[-1.68, 1.67]$	$[-1.24, 1.69]$	$[-1.5, 2.12]$	$[-2.1, 2.77]$
Iris (ours)	$[-397, 397]$	$[-1.55e^3, 1.55e^3]$	$[-2.61, 2.3]$	$[-2.3, 2.84]$	$[-4.4e^3, 4.4e^3]$
Letter (sim)	$[-4.87e^{-5}, 5.46e^{-5}]$	$[-6.47e^{-3}, 7.25e^{-3}]$	$[-1.29, 1.03]$	$[-0.869, 2.21]$	$[-0.0104, 0.0129]$
Letter (ours)	$[-2.84e^{-3}, 2.86e^{-3}]$	$[-0.301, 0.307]$	$[-3.11, 2.02]$	$[-1.87, 3.31]$	$[-1.01, 1.01]$
Credit (sim)	$[-7.11e^{-3}, 0.0115]$	$[-0.0994, 0.0989]$	$[-2.19, 3.9]$	$[-3.89, 3.03]$	$[-0.314, 0.245]$
Credit (ours)	$[-0.606, 0.606]$	$[-32.9, 32.9]$	$[-11.5, 10.7]$	$[-6.25, 5.62]$	$[-206, 206]$
Drive (sim)	$[-1.36e^5, 1.65e^5]$	$[-1.55, 1.39]$	$[-962, 1.01e^3]$	$[-1.01e^3, 970]$	$[-345, 308]$
Drive (ours)	$[-1.4e^{13}, 1.4e^{13}]$	$[-6.56e^{15}, 6.56e^{15}]$	$[-1e^4, 8.36e^3]$	$[-3.42e^3, 3.44e^3]$	$[-2.26e^{19}, 2.26e^{19}]$
	$\boldsymbol{P}_i$	$\boldsymbol{\beta}_i$	$\boldsymbol{e}_i$	$\boldsymbol{h}_i$	$\boldsymbol{y}$
Digits (sim)	$[-0.0544, 0.0705]$	$[-0.351, 0.451]$	$[-10.6, 9.15]$	$[-10.0, 9.19]$	$[-3.16, 3.25]$
Digits (ours)	$[-27.4, 26.2]$	$[-8.93e^4, 8.93e^4]$	$[-23.1, 20.1]$	$[-22.5, 20.8]$	$[-3.39e^7, 3.39e^7]$
Iris (sim)	$[-1.72, 11.4]$	$[-3.44, 5.32]$	$[-2.44, 1.41]$	$[-3.0, 2.21]$	$[-1.23, 1.79]$
Iris (ours)	$[-358, 435]$	$[-4.4e^3, 4.4e^3]$	$[-2.53, 1.58]$	$[-3.1, 2.38]$	$[-1.71e^4, 1.71e^4]$
Letter (sim)	$[-1.66e^{-3}, 2.45e^{-3}]$	$[-0.34, 0.294]$	$[-4.6, 5.33]$	$[-4.86, 6.01]$	$[-1.25, 1.18]$
Letter (ours)	$[-9.2e^{-3}, 0.0126]$	$[-1.35, 0.99]$	$[-6.6, 7.8]$	$[-6.87, 8.48]$	$[-95.7, 95.3]$
Credit (sim)	$[-0.0649, 0.115]$	$[-1.83, 1.38]$	$[-4.66, 5.5]$	$[-5.55, 6.22]$	$[-2.18, 3.77]$
Credit (ours)	$[-0.625, 1.05]$	$[-204, 208]$	$[-8.29, 9.66]$	$[-9.19, 10.4]$	$[-1.09e^4, 1.09e^4]$
Drive (sim)	$[-1.4e^5, 1.7e^5]$	$[-317, 318]$	$[-9.9, 7.42]$	$[-9.35, 8.29]$	$[-1.21e^3, 318]$
Drive (ours)	$[-1.4e^{13}, 1.4e^{13}]$	$[-2.26e^{19}, 2.26e^{19}]$	$[-18.3, 16.8]$	$[-17.7, 16.0]$	$[-1.06e^{22}, 1.06e^{22}]$

Table 3 The "Ops" column shows the total number of arithmetic operations, and the "Overflow/Underflow" column shows the number of overflow or underflows that happened during the experiment. The rate of overflow/underflows is written in ().

	Ops	Overflow/Underflow
Digits (sim)	5,512,688,688	0
Digits (ours)		0
Iris (sim)	4,714,041	197,342 (4.19%)
Iris (ours)		0
Letter (sim)	17,793,216,000	0
Letter (ours)		0
Credit (sim)	11,039,328,000	0
Credit (ours)		0
Drive (sim)	187,259,827,356	5,467,945,469 (2.92%)
Drive (ours)		0

online training samples. ⑤ Generate 250 random training samples $\{\boldsymbol{x}, \boldsymbol{t}\}$ with uniform distribution of $[0, 1]$. ⑥ Feed all the random samples into the training module and the prediction module for each OS-ELM Core and check the number of overflow/underflows that arose. ⑦ Repeat 4-6 until all online training samples run out.

The result is shown in Table 3. The simulation method caused no overflow or underflows in three datasets out of five, however, it suffered from as many overflow/underflows as $2.92 \sim 4.19\%$ in the other two datasets, where a few overflow/underflows arose in an early training step and were propagated to subsequent steps, resulting in a drastic increase in overflow/underflows. This cannot be perfectly prevented as long as a random exploration is taken in interval analysis. The proposed method, on the other hand, encountered totally no overflow or underflows as it analytically derives upper and lower bounds of variables and computes sufficient bit-widths where no overflow or underflows can happen. Although the proposed method produces some redundant bits and it results in a larger area size (see Sect. 5.4), it safely realizes an overflow/underflow-free fixed-point OS-ELM circuit.

5.3 Verification of Hypothesis

Figure 6 shows the entire result of the experiment described in Sect. 3.2.1. We observed similar outcomes to Figure 4 for all the datasets, which supports our hypothesis that $\boldsymbol{A}_i \in \{\boldsymbol{\gamma}_i^{(1)}, \ldots, \boldsymbol{\gamma}_i^{(10)}, \boldsymbol{P}_i, \boldsymbol{\beta}_i, \boldsymbol{e}_i, \boldsymbol{h}_i\}$ roughly satisfies $[\min(\boldsymbol{A}_1), \max(\boldsymbol{A}_1)] \supseteq [\min(\boldsymbol{A}_i), \max(\boldsymbol{A}_i)]$ for $2 \leq i$.

In iterative learning algorithms it is known that learning parameters ($\boldsymbol{\beta}_i$ and $\boldsymbol{P}_i$ in the case of OS-ELM) gradually converge to some values as training proceeds. We consider that this numerical property resulted in the convergence of the dynamic ranges of $\boldsymbol{\beta}_i$ and $\boldsymbol{P}_i$ as observed in Fig. 6, then it tightened the dynamic ranges of other variables (e.g. $\boldsymbol{\gamma}_i^{(1)}, \ldots, \boldsymbol{\gamma}_i^{(10)}$) too, as a side-effect via enormous number of multiplications existing in the OS-ELM algorithm. We plan to investigate the hypothesis either by deriving an analytical proof or using a larger dataset in the future work.

5.4 Area Cost

In this section the proposed method is evaluated in terms

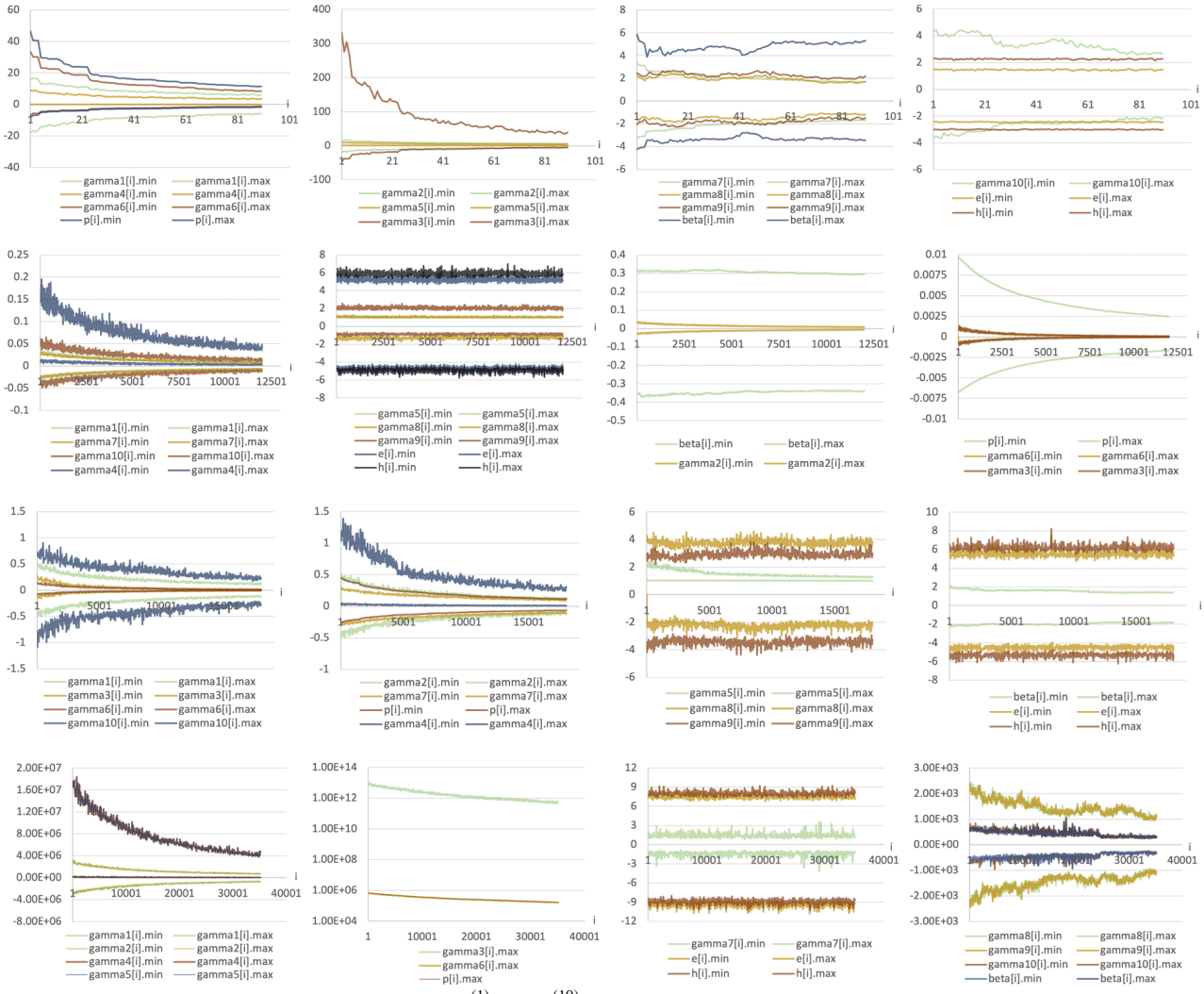


Fig. 6 Observed intervals of $\{\boldsymbol{\gamma}_i^{(1)}, \ldots, \boldsymbol{\gamma}_i^{(10)}, \boldsymbol{P}_i, \boldsymbol{\beta}_i, \boldsymbol{e}_i, \boldsymbol{h}_i\}$ on Iris (top row), Letter (2nd row), Credit (3rd row), and Drive (bottom row), respectively.

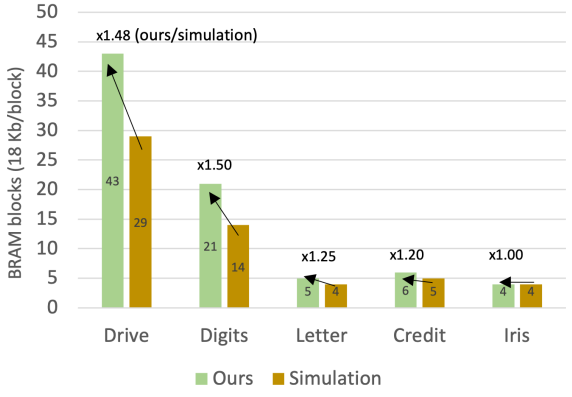


Fig. 7 Comparison of area cost. The green bar represents the BRAM utilization of our method and the brown bar is of the simulation method.

of area cost. We refer to BRAM utilization of OS-ELM Core as "area cost", considering that all the arrays in OS-ELM Core are implemented with BRAM blocks (i.e. the bottleneck of area cost is BRAM utilization). The proposed

Table 4 Notation rules in this paper.

Notation	Description
x (*italic*)	Scaler.
$\hat{x}$	Affine form of x.
$\boldsymbol{x}$ (***bold italic***)	Vector or matrix.
$\hat{\boldsymbol{x}}$	Affine form of $\boldsymbol{x}$ (see Eq. (16) for details).
$x_{[u,v]}$	uv element of $\boldsymbol{x}$.
$\hat{x}_{[u,v]}$	Affine form for the uv element of $\boldsymbol{x}$.
f (upright)	Function (e.g. G, sup, inf, interval).

method is compared with the simulation method introduced in Sect. 5.1 to clarify how much additional area cost arises to guarantee OS-ELM Core being overflow/underflow-free. The experimental procedure is as follows: ① Convert the simulation result into integer bit-widths using Eq. (15) and synthesize OS-ELM Core with the optimized bit-widths. ② Execute the proposed interval analysis method. Convert the result into integer bit-widths and synthesize OS-ELM Core. ③ Check the BRAM utilizations of our method and

Table 5 Description of variables that appear in this paper. The characters used for these variables are the same as the ones used in [2].

Variable	Description
$n, \tilde{N}, m \in \mathbb{N}$	Number of input, hidden, or output nodes of OS-ELM.
$\boldsymbol{\alpha} \in \mathbb{R}^{n \times \tilde{N}}$	Non-trainable weight matrix connecting the input and hidden layers, which is initialized with random values.
$\boldsymbol{\beta} \in \mathbb{R}^{\tilde{N} \times m}$	Trainable weight matrix connecting the hidden and output layers.
$\boldsymbol{P} \in \mathbb{R}^{\tilde{N} \times \tilde{N}}$	Trainable intermediate weight matrix for training $\boldsymbol{\beta}$.
$\boldsymbol{b} \in \mathbb{R}^{1 \times \tilde{N}}$	Non-trainable bias vector of the hidden layer, which is initialized with random values.
G	Activation function applied to the hidden layer output.
$\boldsymbol{x} \in \mathbb{R}^{1 \times n}$	Input vector.
$\boldsymbol{t} \in \mathbb{R}^{1 \times m}$	Target vector.
$\boldsymbol{y} \in \mathbb{R}^{1 \times m}$	Output vector.
$\boldsymbol{h} \in \mathbb{R}^{1 \times \tilde{N}}$	Output vector of the hidden layer (after activation).
$\boldsymbol{e} \in \mathbb{R}^{1 \times \tilde{N}}$	Output vector of the hidden layer (before activation).
$\boldsymbol{X} \in \mathbb{R}^{k \times n}$	Input matrix of batch size = k ($k \in \mathbb{N}$).
$\boldsymbol{T} \in \mathbb{R}^{k \times m}$	Target matrix of batch size = k.
$\boldsymbol{Y} \in \mathbb{R}^{k \times m}$	Output matrix of batch size = k.
$\boldsymbol{H} \in \mathbb{R}^{k \times \tilde{N}}$	Output matrix of the hidden layer with batch size = k (after activation).
$\boldsymbol{\gamma}^{(1)}, \ldots, \boldsymbol{\gamma}^{(10)}$	Intermediate variables that appear in OS-ELM's training algorithm.

the simulation method. ④ Repeat 1-3 for all the datasets.

The experimental result is shown in Fig. 7. Our method requires 1.0x - 1.5x more BRAM blocks to guarantee that OS-ELM Core never encounter overflow and underflow, compared to the simulation method.

Remember that a multiplication in AA causes overestimation of interval; there should be a strong correlation between the additional area cost (i.e. simulation - ours) and the number of multiplications in OS-ELM's training and prediction algorithms.

$$\mathrm{M}(n, \tilde{N}, m) = 4\tilde{N}^2 + (3m + n + 1)\tilde{N} \tag{19}$$

$\mathrm{M}(n, \tilde{N}, m)$ calculates the total number of multiplications in OS-ELM's training and prediction algorithms, where n, $\tilde{N}$, or m is the number of input, hidden, or output nodes, respectively. Equation (19) shows that $\tilde{N}$ has the largest impact on additional area cost, which is consistent with the result that 2.0x more additional area cost was observed in Drive compared to Digits, with fewer inputs nodes (Drive: 48, Digits: 64), more hidden nodes (Drive: 64, Digits: 48), and almost the same number of hidden nodes (Drive: 11, Digits: 10). We conclude that the proposed method is highly effective especially when the model size is small, and that the number of hidden nodes has the strongest impact on additional area cost.

6. Related Work

6.1 Static Interval Analysis for Iterative Algorithms

Existing general-use static interval analysis methods, including AA, deal with iterative algorithms by expanding them into feed-forward computation graphs using loop unrolling [7]–[9], [18]. There must be a termination condition for the target iterative algorithm to apply loop unrolling; otherwise it cannot be represented in a feed-forward computation graph and interval analysis becomes infeasible. OS-ELM's training algorithm has no termination condition and existing methods alone cannot realize interval analysis for OS-ELM.

Kinsman et al. proposed an SMT (satisfiability modulo theory) based static interval analysis framework designed for iterative algorithms [29] and demonstrated that the method worked for famous iterative algorithms, newton-raphson method and conjugate gradient method, however, it still cannot handle algorithms that do not have termination conditions like OS-ELM. Several papers [30]–[34] proposed analytical interval analysis methods for LTI (linear time invariant) circuits with feedback loops which cannot be translated into feed-forward computation graphs, but the methods are dedicated to LTI circuits and not for OS-ELM. As far as we know this paper is the first work to realize a static interval analysis for OS-ELM by leveraging a numerical property of OS-ELM we pointed out in Sect. 3.2.1.

6.2 Division on Static Interval Analysis

Most of static interval analysis methods assume that all denominators within a target algorithm do not take zero [8], [9], [18], or interval analysis becomes infeasible. SMT-based methods proposed by Kinsman et al. provide a mitigation solution to handle a denominator that can take zero by forcibly adding a numerical constraint that prevents it from taking zero (e.g. $y \geq 0.01$ for $z = \frac{x}{y}$), however, the constraint $y \geq 0.01$ is inconsistent with the fact $0 \in y$, which can lead to overestimation or underestimation in subsequent intervals.

We took the safest strategy; we analytically proved that the denominator $\gamma_i^{(5)} = 1 + \boldsymbol{h}_i \boldsymbol{P}_{i-1} \boldsymbol{h}_i^T$ of OS-ELM does not take zero at any $i \geq 1$, and proposed a mathematical trick based on the proof to safely represent OS-ELM in AA. Note that the contributions can apply to not only AA but also other static interval analysis methods.

7. Conclusion

In this paper we proposed an overflow/underflow-free bit-width optimization method for fixed-point OS-ELM digital circuits. In the proposed method affine arithmetic is used to estimate the intervals of intermediate variables and compute the optimal number of integer bits that never cause overflow

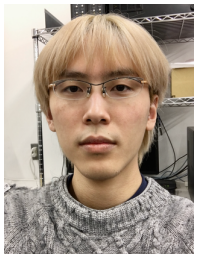
and underflow. We clarified two critical problems in realizing the proposed method: (1) OS-ELM's training algorithm is an iterative algorithm and the computation graph grows endlessly, which makes interval analysis infeasible in affine arithmetic. (2) OS-ELM's training algorithm has a division operation and if the denominator can take zero OS-ELM can not be represented in affine arithmetic.

We proposed an empirical solution to prevent the computation graph from growing endlessly, based on simulation results. We also analytically proved that the denominator does not take zero at any training step, and proposed a mathematical trick based of the proof to safely represent OS-ELM in affine arithmetic. Experimental results confirmed that no underflow/overflow occurred in our method on multiple datasets. Our method realized overflow/underflow-free OS-ELM digital circuits with 1.0x - 1.5x more area cost compared to the baseline simulation method where overflow or underflow can happen.

References

[1] M. Tsukada, M. Kondo, and H. Matsutani, "A neural network-based on-device learning anomaly detector for edge devices," IEEE Trans. Comput., vol.69, no.7, pp.1027–1044, July 2020.
[2] N. Liang, G. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," IEEE Trans. Neural Netw., vol.17, no.6, pp.1411–1423, Nov. 2006.
[3] M. Tsukada, M. Kondo, and H. Matsutani, "OS-ELM-FPGA: An FPGA-based online sequential unsupervised anomaly detector," Proc. International European Conference on Parallel and Distributed Computing Workshops, pp.518–529, Aug. 2018.
[4] J. Villora, A. Muñoz, M. Mompean, J. Aviles, and J. Martinez, "Moving learning machine towards fast real-time applications: A high-speed FPGA-based implementation of the OS-ELM training algorithm," Electronics, vol.7, no.11, pp.1–23, Nov. 2018.
[5] A. Safaei, Q. Wu, T. Akilan, and Y. Yang, "System-on-a-chip (SoC)-based hardware acceleration for an online sequential extreme learning machine (OS-ELM)," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.38, no.11, pp.2127–2138, 2019.
[6] D. Lee, A. Gaffer, R. Cheung, O. Mencer, W. Luk, and G. Constantinides, "Accuracy-guaranteed bit-width optimization," IEEE Trans. Comput.-Aided Design. Integr. Circuits Syst., vol.25, no.10, pp.1990–2000, Oct. 2006.
[7] A. Kinsman and N. Nicolici, "Bit-width allocation for hardware accelerators for scientific computing using SAT-modulo theory," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.29, no.3, pp.405–413, March 2010.
[8] D. Boland and G. Constantinides, "Bounding variable values and round-off effects using Handelman representations," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.30, no.11, pp.1691–1704, Nov. 2011.
[9] J. Stolfi and L. Figueiredo, "Self-validated numerical methods and applications," Brazilian Mathematics Colloquium monographs, July 1997.
[10] G. Huang, Q. Zhu, and C. Siew, "Extreme learning machine: A new learning scheme of feedforward neural networks," Proc. International Joint Conference on Neural Networks, pp.985–990, July 2004.
[11] D. Menard, G. Caffarena, J. Antonio, A. Lopez, D. Novo, and O. Sentieys, "Fixed-point refinement of digital signal processing systems," pp.1–37, The Institution of Engineering and Technology, May 2019.
[12] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens, "A methodology and design environment for DSP ASIC fixed point refinement," Design, Automation and Test in Europe Conference and Exhibition, pp.271–276, March 1999.
[13] A. Gaffar, O. Mencer, and W. Luk, "Unifying bit-width optimisation for fixed-point and floating-point designs," The Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp.79–88, April 2004.
[14] H. Keding, M. Willems, and H. Meyr, "FRIDGE: A fixed-point design and simulation environment," Design, Automation and Test in Europe Conference and Exhibition, pp.429–435, Feb. 1998.
[15] C. Shi and R. Brodersen, "Automated fixed-point data-type optimization tool for signal processing and communication systems," Design Automation Conference, pp.478–483, July 2004.
[16] J. Cong, K. Gururaj, B. Liu, C. Liu, Z. Zhang, S. Zhou, and Y. Zou, "Evaluation of static analysis techniques for fixed-point precision optimization," Proc. IEEE Symposium on Field Programmable Custom Computing Machines, pp.231–234, April 2009.
[17] S. Vakili, J. Langlois, and G. Bois, "Enhanced precision analysis for accuracy-aware bit-width optimization using affine arithmetic," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.32, no.12, pp.1853–1865, Dec. 2013.
[18] R. Moore, "Interval analysis," Science, vol.158, no.3799, pp.365–365, Oct. 1967.
[19] C. Fang, R. Rutenbar, and T. Chen, "Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs," Proc. International Conference on Computer Aided Design, pp.1–8, Nov 2003.
[20] Y. Pu and Y. Ha, "An automated, efficient and static bit-width optimization methodology towards maximum bit-width-to-error tradeoff with affine arithmetic model," Proc. Asia and South Pacific Conference on Design Automation, pp.886–891, Jan. 2006.
[21] S. Wang and X. Qing, "A mixed interval arithmetic/affine arithmetic approach for robust design optimization with interval uncertainty," J. Mechanical Design, vol.138, no.4, pp.041403-1–041403-10, April 2016.
[22] R. Bellal, E. Lamini, H. Belbachir, S. Tagzout, and A. Belouchrani, "Improved affine arithmetic-based precision analysis for polynomial function evaluation," IEEE Trans. Comput., vol.68, no.5, pp.702–712, May 2019.
[23] E. Alpaydin and C. Kaynak, "Optical recognition of handwritten digits data Set," https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits, 1998.
[24] R. Fisher, "Iris data set," http://archive.ics.uci.edu/ml/datasets/Iris/, 1936.
[25] D. Slate, "Letter recognition data set," https://archive.ics.uci.edu/ml/datasets/Letter+Recognition, 1890.
[26] I. Yeh, "Default of credit card," https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients, 2016.
[27] M. Bator, "Sensorless drive diagnosis," https://archive.ics.uci.edu/ml/datasets/dataset+for+sensorless+drive+diagnosis, 2015.
[28] "Digilent PYNQ-Z1," https://japan.xilinx.com/products/boards-and-kits/1-hydd4z.html
[29] A. Kinsman and N. Nicolici, "Automated range and precision bit-width allocation for iterative computations," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.30, no.9, pp.1265–1278, Sept. 2011.
[30] J. Lopez, C. Carreras, and O. Nieto-Taladriz, "Improved interval-based characterization of fixed-point LTI systems with feedback loops," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.26, no.11, pp.1923–1933, Nov. 2007.
[31] O. Sarbishei, Y. Pang, and K. Radecka, "Analysis of range and precision for fixed-point linear arithmetic circuits with feedbacks," Proc. IEEE International High Level Design Validation and Test Workshop, pp.25–32, June 2010.
[32] O. Sarbishei, K. Radecka, and Z. Zilic, "Analytical optimization of bit-widths in fixed-point LTI systems," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.31, no.3, pp.343–355, March 2012.
[33] E. Lamini, R. Bellal, H. Belbachir, and A. Belouchrani, "Enhanced bit-width optimization for linear circuits with feedbacks," Proc. International Design and Test Symposium, pp.168–173, Dec. 2014.

[34] S. Ohno and S. Wang, "Overflow-free realizations for LTI digital filters," Proc. International Symposium on Intelligent Signal Processing and Communication Systems, pp.1–2, Dec. 2019.



Mineto Tsukada received the BE degree from Keio University, Yokohama, Japan in 2018. He is currently a doctoral course student in the Graduate School of Science and Technology, Keio University. His research interests include computer architecture and machine learning.

Hiroki Matsutani received the BA, ME, and PhD degrees from Keio University, Yokohama, Japan in 2004, 2006, and 2008, respectively. He is currently an associate professor in the Department of Information and Computer Science, Keio University. His research interests include the areas of computer architecture.