PAPER Special Section on Low-Power and High-Speed Chips

An Area-Efficient Recurrent Neural Network Core for Unsupervised Time-Series Anomaly Detection*

Takuya SAKUMA[†a)], ***Nonmember and*** **Hiroki MATSUTANI**[†], ***Member***

SUMMARY Since most sensor data depend on each other, time-series anomaly detection is one of practical applications of IoT devices. Such tasks are handled by Recurrent Neural Networks (RNNs) with a feedback structure, such as Long Short Term Memory. However, their learning phase based on Stochastic Gradient Descent (SGD) is computationally expensive for such edge devices. This issue is addressed by executing their learning on high-performance server machines, but it introduces a communication overhead and additional power consumption. On the other hand, Recursive Least-Squares Echo State Network (RLS-ESN) is a simple RNN that can be trained at low cost using the least-squares method rather than SGD. In this paper, we propose its area-efficient hardware implementation for edge devices and adapt it to human activity anomaly detection as an example of interdependent time-series sensor data. The model is implemented in Verilog HDL, synthesized with a 45 nm process technology, and evaluated in terms of the anomaly capability, hardware amount, and performance. The evaluation results demonstrate that the RLS-ESN core with a feedback structure is more robust to hyper parameters than an existing Online Sequential Extreme Learning Machine (OS-ELM) core. It consumes only 1.25 times larger hardware amount and 1.11 times longer latency than the existing OS-ELM core.

key words: *on-device learning, machine learning, anomaly detection*

1. Introduction

In recent years, with the development of IoT devices, we are entering an era in which things communicate not only with people but also with each other. Anomaly detection on sensor data is one of practical IoT applications. For example, monitoring the vibration pattern of fans in data centers can prevent servers from going down. Because the real-world environment changes little by little, their data also change little by little. In other words, their time-series data depend on their previous state. Thus, when using data from IoT devices to detect anomalies, detection models should take into account the dependencies between the data. In addition, IoT devices often provide multi-dimensional data. For example, in the case of human activity recognition, it is necessary to make use of the rotational information obtained from various parts of a human body [1].

Recurrent Neural Networks (RNNs) are good at processing sequential data of arbitrary length because they have an internal feedback structure. This makes them applicable to process time-series data. Typical RNNs use the Stochastic Gradient Descent (SGD) method for learning. It is an iterative method for optimizing target neural networks with suitable smoothness properties, such as differentiable or subdifferentiable. However, this iterative method often takes a long time to achieve high generalization capability. It is computationally expensive to train neural network models with SGD on IoT devices with limited resources [2]. This issue is not a matter if the IoT devices execute only prediction tasks using the model parameters which have been trained on a server with rich computational resources. Such a method is useful if the data is always stationary and the data distribution do not change from the time of learning. In some cases, this assumption may not be practical for IoT devices, because the real world environment often changes from time to time. This phenominon is referred as concept drift [3]. A lightweight approach to train the model directly on the device is thus needed.

Echo State Networks (ESNs) [4] are simple RNNs that can be trained at low computational cost using the least-squares method rather than the SGD. In this paper, we propose its area-efficient hardware implementation for edge devices and adapt it to human activity anomaly detection as an example of interdependent time-series sensor data. [**] The model is implemented in Verilog HDL, synthesized with a 45 nm process technology, and evaluated in terms of the anomaly detection capability, hardware amount, and performance.

The rest of this paper is organized as follows. Section 2 introduces RNNs for time-series data, their lightweight approach, and the scoring algorithm for anomaly determination. Section 3 shows related work. Section 4 proposes an area-efficient RNN core with anomaly determination logic, and Sect. 5 describes the RTL implementation. Section 6 shows the evaluation results. Section 7 concludes this paper.

2. Background

An anomaly detection in time-series data consists of two steps. The first step is time-series prediction, where the model predicts the future inputs. The second step is anomaly determination. In the anomaly detection in time-series data, an output value of the first step is compared with the actual

Manuscript received July 3, 2020.
Manuscript revised October 12, 2020.
Manuscript publicized December 15, 2020.

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

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

a) E-mail: sakuma@arc.ics.keio.ac.jp

DOI: 10.1587/transele.2020LHP0003

[**]This paper is an extended version of our 3-page extended abstract appeared in [5].

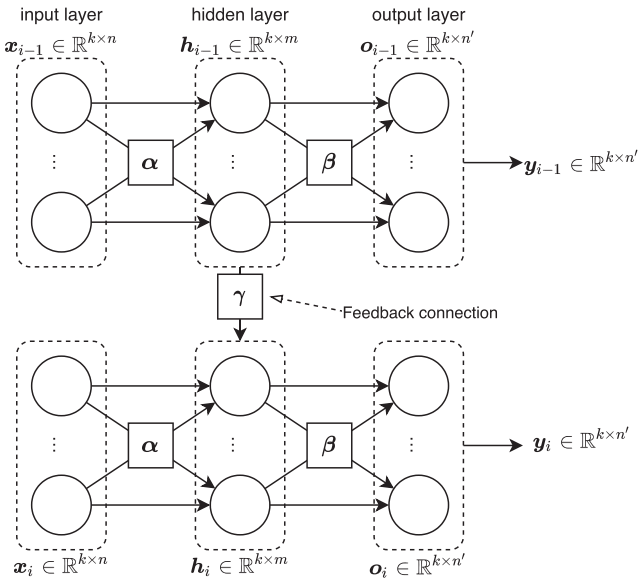


Fig. 1 Recurrent neural network

input value. When the model gets normal input data, the error value becomes small because the model can accurately predict the answer.

In this section, we explain the RNNs which are often used for time-series prediction and their lightweight approach. Then, we introduce Hotelling's T-square test [6] as one of the anomaly determination methods.

2.1 Recurrent Neural Networks

RNNs are one of the neural networks for processing time-series data. The input data to RNNs are defined as follows. Consider a time-series

$$X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_\tau}\}, \tag{1}$$

where each sample $\boldsymbol{x_i} \in \mathbb{R}^{k\times n}$ in the time-series is an n-dimensional vector of batch size $k \in \mathbb{N}$. RNNs predict l outputs for d inputs. d is called the window size. We assume d is 1 unless otherwise specified.

The recursive connection plays the role of a dynamic memory, so that RNNs can reflect task demands in the context of prior internal states [7]. Although there are various RNNs structures, this paper deals with the one that generates output data at each time step and has a feedback connection between hidden units [8], as shown in Fig. 1.

RNNs start prediction or learning by making the initialization state $\boldsymbol{h_0} \in \mathbb{R}^{k\times m}$ in the forward propagation; then at each time step from $i = 1$ to $i = \tau$, the model calculates an output $\boldsymbol{y_i} \in \mathbb{R}^{k\times n'}$ with the following formula:

$$\boldsymbol{h_i} = \boldsymbol{G}(\boldsymbol{\alpha x_i} + \boldsymbol{\gamma h_{i-1}}), \tag{2}$$

$$\boldsymbol{y_i} = \boldsymbol{F}(\boldsymbol{\beta h_i}), \tag{3}$$

where $\boldsymbol{G}$ and $\boldsymbol{F}$ are activation functions for the hidden and output layers. $\boldsymbol{\alpha} \in \mathbb{R}^{m\times n}$, $\boldsymbol{\beta} \in \mathbb{R}^{n'\times m}$, and $\boldsymbol{\gamma} \in \mathbb{R}^{m\times m}$ are the weight matrices of the connections from the input to the hidden layer, from the hidden layer to the output layer, and from the hidden layer to the hidden layer, respectively. Typically, the output $\boldsymbol{y_i}$ derived from an RNN model is connected to an input for another model. Stacking them, the model improves the representational capability.

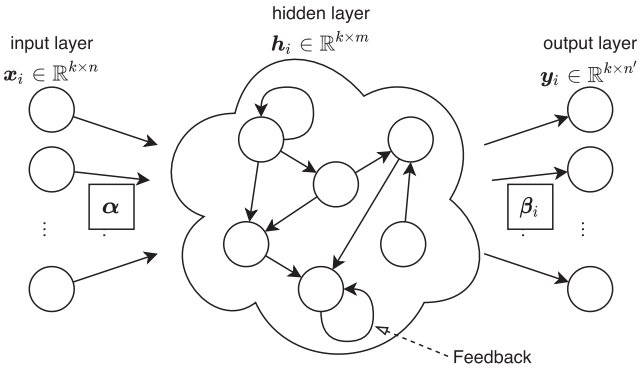


Fig. 2 Echo state network

As shown in Eq. (4), a typical RNN is optimized by the iterative method of SGD because of non-linearity. For the loss value $l_i \in \mathbb{R}$ obtained by comparing the model's prediction result $\boldsymbol{y_i}$ with the correct answer data $\boldsymbol{t_i}$, the model parameters $\boldsymbol{w} \in \{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$ are updated by the following equation:

$$\boldsymbol{w} = \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} l_i(\boldsymbol{w}), \tag{4}$$

where $\nabla_{\boldsymbol{w}}$ is a vector differential operator for $\boldsymbol{w}$, and η is a learning rate. l_i is a loss value, which can be calculated with the following equation:

$$l_i = \frac{1}{n'} \|\boldsymbol{y_i} - \boldsymbol{t_i}\|_2^2. \tag{5}$$

By iterating Eq. (4) and updating the parameters, the model can obtain optimal prediction results for the training dataset. This method can optimize arbitrary neural network models with suitable smoothness properties. However, it requires a large data set and a long computation time to achieve a high generalization. The delay of learning widens the gap between the latest true distributions of normal data and ones trained by the models [9], which makes detecting anomalies more difficult. To solve this issue, the iterative approach should be replaced with the approach that sequentially updates the parameters. In the following subsections, we describe an example of this solution.

2.2 Echo State Networks

An ESN [4] is a simple approach to train RNN. As shown in Fig. 2, it consists of input layer, hidden layer, and output layer.

Given an i-th input $\boldsymbol{x_i}$, the model calculates the hidden layer output $\boldsymbol{h_i}$ as follows:

$$\boldsymbol{h_i} = \boldsymbol{G}\Big((1-\delta)\boldsymbol{h_{i-1}} + \delta(\boldsymbol{x_i}\boldsymbol{\alpha} + \boldsymbol{h_{i-1}}\boldsymbol{\gamma})\Big), \tag{6}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{\gamma} \in \mathbb{R}^{m\times m}$ are fixed weights, $\delta \in \mathbb{R}$ is a leak rate, and $\boldsymbol{G}$ is an activation function, respectively. The

larger δ is, the less the effect of input data $\boldsymbol{x}_i$, because the impact of the feedback becomes stronger, and the model will keep the previous state. Then, the model outputs $\boldsymbol{y_i} \equiv \boldsymbol{h_i}\boldsymbol{\beta_i}$. Here, $\boldsymbol{\beta_i} \in \mathbb{R}^{m\times n'}$ is the i-th weight connecting the hidden and output layers updated by the least-squares method. Using Mean Squared Error, the objective function is defined as follows:

$$L(\boldsymbol{\beta_i}) = \frac{1}{2}\|\boldsymbol{h_i}\boldsymbol{\beta_i} - \boldsymbol{t_i}\|_2^2. \tag{7}$$

By minimizing L for $\boldsymbol{\beta_i}$, the optimal solution is obtained by the following equation:

$$\boldsymbol{\beta_{i+1}} = \boldsymbol{h_i^+}\boldsymbol{t_i}, \tag{8}$$

where $\boldsymbol{h_i^+}$ is the pseudo-inverse matrix of $\boldsymbol{h_i}$ and can be calculated using well-known methods, such as Singular Value Decomposition and QR Decomposition.

For updating $\boldsymbol{\beta_i}$ adaptively, we can use the Recursive Least-Squares (RLS) [10] algorithm known as a fast online adaptation method in linear systems.

2.3 Recursive Least-Squares Echo State Network

An ESN with the RLS based model update algorithm is known as RLS-ESN [11]. RLS is an algorithm that sequentially applies the least-squares method. We consider the case in which the i-th data of batch size k is given as an input data. RLS with forgetting rate λ for Eq. (7) is represented as follows:

$$L(\boldsymbol{\beta_i}) = \frac{1}{2}\left\|\begin{bmatrix}\lambda^{i-1}\\ \vdots \\ 1\end{bmatrix}\left(\begin{bmatrix}\boldsymbol{h_1}\\ \vdots \\ \boldsymbol{h_i}\end{bmatrix}\boldsymbol{\beta_i} - \begin{bmatrix}\boldsymbol{t_1}\\ \vdots \\ \boldsymbol{t_i}\end{bmatrix}\right)\right\|_2^2. \tag{9}$$

Here, using $\boldsymbol{k_i} \equiv \begin{bmatrix}\lambda^{i-1}\boldsymbol{h_1}\\ \vdots \\ \boldsymbol{h_i}\end{bmatrix}^T \begin{bmatrix}\boldsymbol{h_1}\\ \vdots \\ \boldsymbol{h_i}\end{bmatrix}$, the optimal $\boldsymbol{\beta_i}$ that minimizes $L(\boldsymbol{\beta_i})$ is expressed as follows:

$$\boldsymbol{\beta_i} = \boldsymbol{\beta_{i-1}} + \boldsymbol{k_i^{-1}}\boldsymbol{h_i^T}(\boldsymbol{t_i} - \boldsymbol{h_i}\boldsymbol{\beta_{i-1}}) \tag{10}$$

$$\boldsymbol{k_i} = \lambda\boldsymbol{k_{i-1}} + \boldsymbol{h_i^T}\boldsymbol{h_i}. \tag{11}$$

Assuming $\boldsymbol{p_i} \equiv \boldsymbol{k_i^{-1}}$ and $\boldsymbol{q_i} \equiv \frac{1}{\lambda}\boldsymbol{p_i}$, Eqs. (10) and (11) finally yield the following equations:

$$\boldsymbol{p_i} = \boldsymbol{q_{i-1}} - \boldsymbol{q_{i-1}}\boldsymbol{h_i^T}(\boldsymbol{I} + \boldsymbol{h_i}\boldsymbol{q_{i-1}}\boldsymbol{h_i^T})^{-1}\boldsymbol{h_i}\boldsymbol{q_{i-1}} \tag{12}$$

$$\boldsymbol{\beta_i} = \boldsymbol{\beta_{i-1}} + \boldsymbol{q_i}\boldsymbol{h_i^T}(\boldsymbol{t_i} - \boldsymbol{h_i}\boldsymbol{\beta_{i-1}}) \tag{13}$$

As shown in Eqs. (12) and (13), the weight parameter $\boldsymbol{\beta_i}$ is derived from the previous intermediate result $\boldsymbol{p_{i-1}}$. Therefore, the dedicated memory or storage for storing past training data is unnecessary, and thus the requirements for sequential learning are satisfied.

2.4 Hotelling's T-Square Test

Hotelling's T-square test [6] is a method to judge whether two populations are coinciding or not. In anomaly detection, the probability distribution of the input data is compared to that of the normal data in order to see whether the input data is normal or not. Assume that the loss values $D = \{l_1, l_2, \ldots, l_\tau\}$ independently follow the normal distribution as follows:

$$N(l_i|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma}}\exp\left(-\frac{(l_i-\mu)^2}{2\sigma}\right), \tag{14}$$

where μ is the mean and σ^2 is the variance. Assuming the dataset contains a few or no anomalous data, the anomaly score a_i of the loss value l_i is expressed as follows:

$$a_i = \frac{(l_i-\mu)^2}{\sigma^2}. \tag{15}$$

When the number of samples τ is sufficiently large, a_i follows a χ-square distribution with 1-degree of freedom and a scale factor of 1:

$$\chi^2(a_i|1,1) = \frac{1}{2\Gamma(1/2)}\left(\frac{a_i}{2}\right)^{-1/2}\exp\left(-\frac{a_i}{2}\right), \tag{16}$$

where Γ is gamma function, where

$$\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt. \tag{17}$$

The integral from 0 to a_i of χ-square distribution is

$$\int_0^{a_i}\chi^2(x|1,1)dx = 1 - r, \tag{18}$$

where $r(0 \le r \le 1)$ is a confidence interval. By determining r, we automatically get a threshold. For example, when the confidence interval r is set to 0.99, the threshold a^* is

$$\int_0^{a^*}\chi^2(x|1,1)dx = 0.01 \tag{19}$$

$$\therefore a^* \simeq 7.88.$$

3. Related Work

In [1], anomaly detection of time-series data is addressed using Long Short Term Memory (LSTM) [12], which is a type of RNN as well as ESN. As a network architecture, they use stacked LSTM networks for anomaly detection in time-series data. The stacked LSTM consists of two LSTM models and connects one output to the other input. The network is trained with normal data in advance and used as a predictor over a number of time steps. The loss values are modeled as a multivariate Gaussian distribution with pre-trained parameters, which is used to assess the likelihood of anomalous behavior. As compared by [13], LSTM must spend more training time than ESN unless there are too many parameters to optimize; thus, the stacked LSTMs are generally computationally expensive if we sequentially optimize their parameters on IoT devices.

There is a study of models that can be trained on low-end edge devices, such as IoT devices. In [2], a low-cost and low-latency on-device learning design using Online Sequential Extreme Learning Machine (OS-ELM) [14] is proposed and implemented on FPGAs.

OS-ELM is a neural network similar to RLS-ESN that is trained sequentially with RLS. Unlike RLS-ESN, the hidden layer in OS-ELM is calculated by the following equation:

$$\boldsymbol{h_i} = \boldsymbol{G}(\boldsymbol{x_i}\boldsymbol{\alpha}). \tag{20}$$

To speed up the calculations, the authors of [2] propose a hardware implementation by limiting the batch size k to 1. The computational cost of the matrix inversion $(\boldsymbol{I} + \boldsymbol{h_i}\boldsymbol{q_{i-1}}\boldsymbol{h_i^T})^{-1}$ in Eq. (13) is significantly reduced because the size of the target matrix $(\boldsymbol{I} + \boldsymbol{h_i}\boldsymbol{q_{i-1}}\boldsymbol{h_i^T})$ is $k \times k$. As a result, parameter updating algorithm derived from Eq. (13) becomes as follows:

$$\boldsymbol{p_i} = \boldsymbol{q_{i-1}} - \frac{\boldsymbol{q_{i-1}}\boldsymbol{h_i^T}\boldsymbol{h_i}\boldsymbol{q_{i-1}}}{\boldsymbol{I} + \boldsymbol{h_i}\boldsymbol{q_{i-1}}\boldsymbol{h_i^T}} \tag{21}$$

OS-ELM is not suitable for handling time-series data because it has no feedback structures as shown in Eq. (20). In this paper, we apply the Eq. (21) to RLS-ESN to deal efficiently with time-series data.

FORCE learning [15] is one of the most popular methods for online learning on ESN. In FORCE learning, the weight matrix $\boldsymbol{\beta}$ is updated by RLS as with RLS-ESN [11], but the original FORCE learning [15] has no forgetting structure. As shown in the experiment in Sect. 6.5, the forgetting rate λ causes an enormous impact on the AUC score. Although this paper focuses on RLS-ESN since it was proposed in 2003, there is a similarity between them, and so our approach would be extended to FORCE learning as a future work.

Hardware implementation of echo state network has been studied, but most of them are trained offline [16], [17]. In [18], an implementation of the ESN architecture of reservoir computing with real-time training on FPGA without any software support is shown. However, the model must store an enough data to update weight matrix $\boldsymbol{\beta}$ to follow the concept drift since [18] uses least-squares method for the training algorithm. It is not area-efficient because it requires as much memory as the data for the update.

4. RLS-ESN Core Design

In this section, we propose the RLS-ESN core design that consists of two stages: learning/prediction stage with RLS-ESN and scoring stage with the adaptive Hotelling's T-square test.

4.1 Learning/Prediction Stage with RLS-ESN

We assume both the learning and prediction are executed on edge devices as well as [2]. Communication cost between client and server is a crucial issue especially for sensor devices with limited battery capacity and/or those with limited wireless connection. We propose an approach to make the computational complexity of learning and prediction as small as possible.

We first compare RLS with the SGD which is commonly used to train neural network models. Although we can train deep learning models with an arbitrary number of networks by using SGD, here we consider only updating the final layer of the model. The number of the weight parameters is m^2, assuming that the dimension of the hidden layer and the output layer are m. Since SGD newly computes each weight parameter for each iteration, the time complexity increases proportionally with the number of weight parameters (i.e., $O(m^2)$). Since the model repeats this process to achieve generalization, the time complexity of SGD is $O(nm^2)$ as a result, where n is the number of iterations. The number of iterations depends on the task, but its impact on the complexity cannot be ignored. On the other hand, the time complexity of RLS is $O(m^2)$ from Eqs. (21) and (12) [†]; thus, RLS is superior in terms of computational complexity. Taking advantage of this feature, we adopt RLS-ESN as an RNN to run on edge devices.

We found that computations for the hidden layer contain a lot of constant operations. When dividing the formula of the hidden layer into the constants and variables, they can be transformed as follow.

$$\begin{aligned}
&G\Big((1-\delta)\,\boldsymbol{H_{i-1}} + \delta\,(\boldsymbol{x_i}\boldsymbol{\alpha} + \boldsymbol{H_{i-1}}\boldsymbol{\gamma})\Big)\\
=&G\Big(\boldsymbol{H_{i-1}}\big((1-\delta)\,\boldsymbol{I} + \delta\boldsymbol{\gamma}\big) + \boldsymbol{x_i}\delta\boldsymbol{\alpha}\Big)\\
=&G\left(\left[\frac{\delta\boldsymbol{\alpha}}{(1-\delta)\boldsymbol{I} + \delta\gamma}\right]\Big[\ \boldsymbol{x_i}\ \Big|\ \boldsymbol{H_{i-1}}\ \Big]\right)\\
=&G(\boldsymbol{\zeta}\boldsymbol{z}),
\end{aligned} \tag{22}$$

where the vertical and horizontal bars denote concatenations of two matrices, $\boldsymbol{\zeta} \equiv \left[\frac{\delta\boldsymbol{\alpha}}{(1-\delta)\boldsymbol{I} + \delta\gamma}\right]$, and $\boldsymbol{z} \equiv \Big[\ \boldsymbol{x_i}\ \Big|\ \boldsymbol{H_{i-1}}\ \Big]$. Equation (22) can be represented as a matrix product. The hidden layer can be computed with only a single matrix calculation and activation, simplifying the hardware implementation of the model. After all, an ESN is the equivalent of fully-connected neural networks.

We adopt fixed-point numbers for the RLS-ESN core. RLS-ESN is a neural network that trains and predicts at the same time. In general, to train a neural network, it is better to prepare as many decimal digits as possible, while it is better to implement as few bits as possible to reduce the hardware amount. We analyze a dataset used in Sect. 6 in the software simulation, and then a 35-bit Q20 fixed-point number format is required in our RLS-ESN core for the dataset. Please note that the format should be determined by considering application requirements in terms of cost and accuracy.

[†]The terms with the highest time complexity in Eqs. (21) and (12) are the products of m-dimensional vectors such as $\boldsymbol{q_{i-1}}\boldsymbol{h_i^T}$.

4.2 Scoring Stage with Adaptive Hotelling's T-Square Test

Hotelling's T-square test uses the mean and variance of the normal population. Practically, these values are constants, and they should be determined based on prior knowledge. In other words, it assumes implicit pre-training for them and the stationarity of the data. Data from IoT devices, however, is not always stationary since the real world environment may shift as time goes by. This paper tackles this issue with the following two techniques: 1) calculating weighted statistics μ_i and σ_i^2 using forgetting rate r, and 2) rearranging the equation to be an asymptotic expression to handle sequential input data.

The equations for the mean and variance using the forgetting rate r are as follows:

$$\mu_i = \frac{1}{s_i}\sum_{k=1}^{i} r^{i-k} l_i, \tag{23}$$

$$\sigma_i^2 = \frac{1}{s_i}\sum_{k=1}^{i} r^{i-k}(l_k - \mu_i)^2, \tag{24}$$

where $s_i = \sum_{k=1}^{i} r^{i-k}$. By multiplying the forgetting rate r which is less than one, the older the data becomes, the smaller its effect is. We can then rewrite Eqs. (23) and (24) as follows:

$$\mu_i = \frac{1}{s_i}(r\mu_{i-1}s_{i-1} + l_i) \tag{25}$$

$$\sigma_i^2 = \frac{1}{s_i}\left(r(\mu_{i-1}^2 + \sigma_{i-1}^2)s_i + l_i^2\right). \tag{26}$$

As shown in Eqs. (25) and (26), we can remove the summation and calculate the statistics μ_i and σ_i^2 sequentially.

5. Implementation

The implemented RLS-ESN core shown in Fig. 3 processes both the learning/prediction and the scoring stages.

5.1 Learning/Prediction Stage with RLS-ESN

As mentioned in Sect. 4.2, we adopt the 35-bit Q20 number as a fixed-point number format.

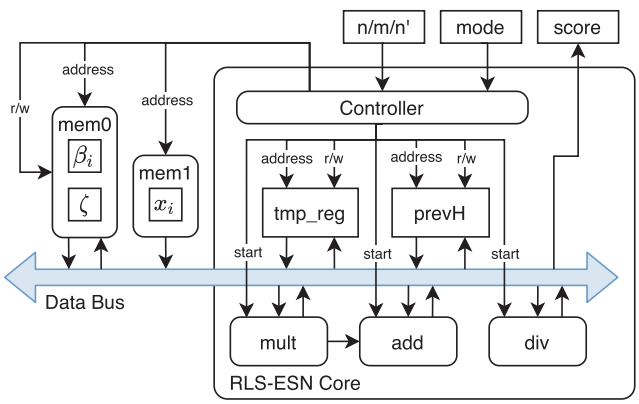


Fig. 3 Overview of RLS-ESN core

Figure 4 illustrates a state transition diagram of the proposed RLS-ESN core. It shows that each circle is corresponding to a matrix operation (e.g., *add*, *mult*, and *div*), and each operation consists of the following three substeps: 1) update the array index, 2) read operands from target matrices and store them in registers, and 3) execute the operation and save the computation result. For multiply-add operation, we use *mult* and *add* in order.

Figure 3 shows a schematic diagram of the implemented RLS-ESN core. The weight parameters of all the layers and input data are stored in *mem*0 and *mem*1, respectively. *mode* signal specifies the operation of the stage (i.e., predict or train). The stage calculates anomaly score a_i (Eq. (15)) in both the modes. It updates the parameter $\boldsymbol{\beta_i}$ only when *mode* is 1. *prevH* is the register that stores the previous-hidden layer and updated with the previous calculation result in parallel. The overall processing flow is as follows:

1. read $\boldsymbol{\zeta}$ from *mem*0 and input data $\boldsymbol{x_i}$ from *mem*1, and feed them to the stage,
2. calculate the hidden layer by using the multiply-add unit,
3. read $\boldsymbol{\beta_i}$ from *mem*0 and compute output and loss values, and
4. calculate the anomaly score a_i using Algorithm 1 which will be explained in Sect. 5.2.

If the *mode* is 1, it updates the weight parameter $\boldsymbol{\beta_i}$; otherwise, it finishes the process.

5.2 Scoring Stage with Adaptive Hotelling's T-Square Test

To reduce hardware amount, arithmetic units used in the ESN should be reused for the adaptive Hotelling's T-squared test stage. In this case, since a fixed-point number format is used for the arithmetic units, a special care is required for avoiding the value overflow.

Regarding the value overflow, we conducted extensive RTL simulations to detect where the value overflows occur in our design. As stated in Sect. 6, we set the forgetting rate to 0.9999, and then intermediate value s_i in Eqs. (25) and (26) becomes about 10000 at $i \to \infty$, resulting in an overflow unless the format has over 15 bits for the integer part. Since the loss value, which is the input to this stage, would become high for anomalous data, we must pay attention to the squared operation in Eqs. (25) and (26). On the other hand, its mean value is much smaller than the maximum loss value, because anomalies occur occasionally.

As for the nature of the arithmetic unit, we need to rearrange the computational steps in order to take advantage of their characteristics. The multiply-add unit in ESN calculates scalar sum and product. We rearrange Eqs. (25) and (26) to utilize this operation as much as possible. Algorithm 1 shows computational steps suitable for the RLS-ESN core.

By dividing s_i and l_i in advance by s_{i-1}, which tend to be relatively large in value, we make the input value to the

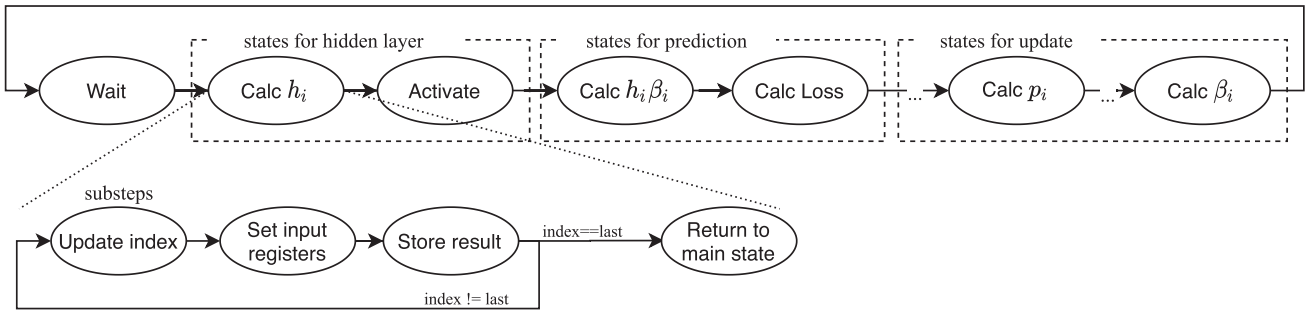


Fig. 4 State transition diagram of RLS-ESN

Algorithm 1 Scoring algorithm with adaptive Hotelling's T-square test

$r \in [0, 1]$
$s_0 \Leftarrow 0$
$\mu_0 \Leftarrow 0$
$\sigma_0^2 \Leftarrow 0$
for until l_i exists **do**
  $s_i \Leftarrow rs_{i-1} + 1$
  $t \Leftarrow r\frac{s_{i-1}}{s_i}$
  $u \Leftarrow \frac{l_i}{s_i}$
  $\mu_i \Leftarrow t\mu_{i-1} + u$
  $v \Leftarrow \mu_i\mu_i + \sigma_{i-1}^2$
  $w \Leftarrow tv - \mu_i$
  $\sigma_i^2 \Leftarrow l_i u + w$
  $a_i \Leftarrow \frac{(l_i-\mu_i)^2}{\sigma_i^2}$
end for

multiplier as small as possible. All the calculation steps of the statistics are in the form of $A \times B + C$, so that we can utilize the multiply-add unit in ESN efficiently in Fig. 3.

6. Evaluations

In this section, we compare the accuracy of RLS-ESN and other anomaly detection algorithms based on neural networks.

We evaluate the anomaly detection capacity at the algorithmic level and the area and performance when implemented as dedicated circuits. The software implementation is used for the former and the RTL design is used for both. We implement the design in Verilog HDL with a 45 nm process technology for the dedicated circuit. Tables 1 and 2 show the details of the experimental environments for the software and hardware implementations. As shown in Table 1, the software environment for LSTM-AD is quite rich because LSTM-AD has a much larger number of parameters than the other models, as we will show in Table 3 (see trainable parameters), and uses SGD for training, which requires abundant computational resources. However, LSTM-AD has an overwhelmingly long training time even in such a rich environment, as we describe the details in Sect. 6.9. We are planning to implement the proposed RLS-ESN core as a dedicated circuit integrated with a sensor at a low frequency. In this experiment, however, the design is operated at 100 MHz to reasonably compare its performance with those running on an embedded CPU at 650 MHz.

6.1 Parameters Selection for RLS-ESN

As shown in Eq. (22), $\boldsymbol{h}_i$ consists of three parameters $\boldsymbol{\alpha}$, $\boldsymbol{\gamma}$, and δ. In this section, we explain the way to generate these parameters. First, we generate $\boldsymbol{\alpha}$ following with [19]. In [19], each element of $\boldsymbol{\alpha}$ is set to either c or $-c$ with equal probability, where c is a real number. The paper shows that the performance tends to improve as c decreases. Then, each element of $\boldsymbol{\gamma}$ satisfies the echo state property [4]. The echo state property is a condition for the ESN to perform well on a given task. Briefly, $\boldsymbol{\gamma}$ should have a fading memory. To satisfy the echo state property, the spectral radius of the $\boldsymbol{h}_i$ should be close to, but not equal to one [4]. The spectral radius works the same way as the forgetting rate for $\boldsymbol{\beta}$. Thus, we regard them as the same parameter. Finally, δ is selected as the one with the highest AUC score throughout the experiments. As mentioned in the paper, the larger δ is, the less the effect of input. We use the highest performing parameters as shown in Sect. 6.7 (i.e., $c = 0.05$ and $\delta = 0.5$).

6.2 Comparison Targets

We compare three kinds of models: RLS-ESN, LSTM-AD, and OS-ELM. Regarding the accuracy of each model, we show that a feedback structure is important for prediction on time-series data. For reference, we implement LSTM-AD based on the previous work [1], which consists of two layers of LSTM-AD. Figure 5 shows the details of the structure of LSTM-AD. Both the layers are fully-connected and their Dropout rate is 0.2. We implement RLS-ESN and OS-ELM algorithms with Numpy 1.16.2 and LSTM-AD with Tensorflow 1.14.0 as the software implementation. Table 3 shows the comparison targets and their parameters. The input data size of RLS-ESN and OS-ELM is the same, but the number of input layer nodes of RLS-ESN is practically 56, because a feedback from the hidden layer in the previous step is also fed to the input layer. The large number of parameters directly leads to an increase in the required RAM amount since both SGD and RLS must temporarily store their corresponding data (i.e., gradients or $\boldsymbol{p}_i$) depending on the number of trainable parameters. Please note that the number of trainable parameters of LSTM-AD is much

Table 1 Environment for software implementation

	For LSTM-AD	For Others
Processor	Intel(R) Core(TM) i7-6850K	Cortex-A9 processor
Operating frequency	3.6GHz	650MHz
Operation system	Ubuntu 18.04	Ubuntu 18.04
Accelerator	GeForce GTX 1080	-

Table 2 Environment for hardware implementation

	Environment
Logic simulator	Icarus Verilog 10.1
Logic synthesis	Synopsys Design Compiler 2018.06-SP4
Technology library	Silvaco 45 nm Open Cell Library
Operation frequency	100 MHz

Table 3 Parameters of each model

	LSTM-AD	OS-ELM	RLS-ESN
Window size d	128	1	1
Forgetting rate r	0.9999	0.9999	0.9999
Input layer size n	3,584	28	28
Hidden layer size m	-	28	28
Output layer size n'	-	28	28
Forgetting rate λ	-	0.9999	0.9999
Trainable parameters	45,948	784	784

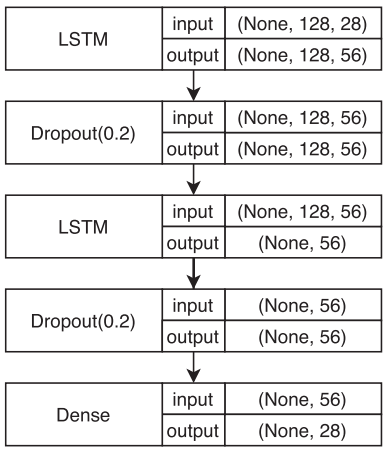


Fig. 5 Details of the structure of LSTM-AD

greater than the others. We use these parameters unless otherwise specified. Please note that, the proposed RLS-ESN core includes the scoring module with adaptive Hotelling's T-Square test, while the existing OS-ELM core [2] does not include it.

6.3 Dataset

We use RealWorld (HAR) dataset in [20] to evaluate detection accuracy in a real-world environment. Fifteen subjects with age 31.9±12.4, height 173.1±6.9 cm, and weight 74.1±13.8 kg perform the following eight actions: descending stairs, climbing stairs, jumping, lying down, standing, sitting, running, and walking. Each input data is a collections of sensor values from smartphones attached to seven different body parts: chest, forearm head, shin thigh, upper arm, and hip.

6.4 Data Preparation

The dataset contains raw data obtained from wearable sensor devices (e.g., accelerometers and gyroscopes). Theoretically, an prediction model can be constructed with raw data, but it is practically difficult for shallow and simple neural networks to acquire a good prediction model by learning only with raw data in our experiment. The model can judge anomalies by whether a subject has made a different motion than before. As a preprocessing step, rotation data is constructed from the raw data. Then, the rotation data is fed to the neural networks so that they can effectively construct a beneficial prediction model even with shallow neural networks. The origin of human activity is the rotation of each human joint. We can easily convert the raw data into rotational data by using well-known methods for extracting rotation from various sensor devices, such as Kalman filter [21] and Madgwick filter [22].

As the dataset contains raw data from accelerometers, gyroscopes, and magnetic field sensors, we adopt the Madgwick filter [22], which uses them to reconstruct the rotation. To restore rotation, the Madgwick filter requires all of these raw data generated at the same time, though the sampling interval of each sensor device is slightly different.

We thus approximate the data at the same timing with bilinear interpolation between the data based on the sampling interval of each sensor device. The function that linearly interpolates the values a and b with the ratio $c \in [0, 1]$ is

$$lerp(a, b, c) = ca + (1 - c)b. \tag{27}$$

We use continuous running data from subjects 1 and 10-13. Although RLS-ESN and OS-ELM do not require pre-training, in order to match the experiment conditions with LSTM-AD, the data of subjects 13 and others are used for test and pre-training, respectively. The number of total steps τ in these data are 30,794 and 124,125, respectively. Given the input data $X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_{\tau-1}}\}$, the correct answer to the model is $Y = \{\boldsymbol{x_2}, \boldsymbol{x_3}, \ldots, \boldsymbol{x_\tau}\}$, where x_i is the i-th sample data, each of which consists of 28 values (i.e., four-dimensional rotation data from seven body parts).

6.5 Accuracy of RLS-ESN Core

We use Area Under Curve (AUC) as a metric to evaluate accuracy. AUC is one of widely-used metrics in anomaly detection task. The false positive rate and the true positive rate are taken as X and Y axes. The respective ratios at various

threshold values are plotted to draw Receiver Operatorating Characteristic (ROC) curve. AUC is then calculated by integrating the ROC curve.

Table 4 shows the AUC scores for each model with different λ parameter. λ indicates how $\boldsymbol{\beta_i}$ tends to forget past data; $\boldsymbol{\beta_i}$ never forgets past information when $\lambda = 1.0000$. All the AUC scores shown in Table 4 are calculated using the anomaly scores a_i from Hotelling's T-square test in Eq. (15). LSTM-AD, RLS-ESN ($\lambda = 0.9999$), and OS-ELM ($\lambda = 0.9999$) achieve higher AUC scores than 94%, while OS-ELM ($\lambda = 0.9990$) result in about 66%. More specifically, the results of OS-ELM and RLS-ESN become comparable to that of LSTM-AD by setting the hyperparameter λ appropriately; however, the AUC score gets worse depending on λ in OS-ELM. Since the OS-ELM model cannot use past data to make time-series predictions, it must learn the relationship between inputs and outputs in order to produce a correct answer without context. In OS-ELM and RLS-ESN, the AUC score decreases to about 76% with $\lambda = 1.0000$. Adapting all previous data means that the new data has a relatively small effect on the model, so that the model cannot deal with the concept drift. Conversely, setting λ to a value less than 1, RLS-ESN has comparable capabilities to LSTM-AD while the additional cost for RLS-ESN core is reasonable as evaluated in Sect. 6.8.

Hardware implementation of RLS-ESN (denoted as RLS-ESN core) in Table 4 achieves a comparable AUC score to that of the software implementation. The hardware implementation has a slightly lower score than the software implementation because, for the hardware implementation, we use the fixed-point number format, which reduces computational accuracy.

Table 4 AUC score of each model

Model	Forgetting rate λ	AUC score [%]
LSTM-AD	-	94.72
OS-ELM	0.9990	65.76
OS-ELM	0.9999	94.73
OS-ELM	1.0000	76.10
RLS-ESN	0.9990	94.55
RLS-ESN	0.9999	95.00
RLS-ESN	1.0000	76.31
RLS-ESN core	0.9999	94.54

6.6 Forgetting Rate for Statistics vs. Accuracy

In this section, we evaluate the anomaly detection accuracy by changing the forgetting rate r for Hotelling's T-square test. If the model is not stable, we may not be able to evaluate the effectiveness of the model correctly by using a fixed threshold. For example, if the anomaly score gradually decreases, it is not appropriate to continue using the threshold. Therefore we demonstrate raw anomaly scores to observe if it tends to be less responsive to anomalies. Figure 6 shows the relationship between the raw anomaly score and the time step for three forgetting rates (e.g., $r = 1.0000$, $r = 0.9999$, and $r = 0.9990$). In the graph, the cyan and red ranges mean normal and abnormal, respectively. With $r = 1.0000$, the variance σ_i^2 increases due to the deviation of the distribution from the initial training data, and so the model becomes less responsive for abnormal inputs. On the other hand, in the case of $r = 0.9999$, the influence of past data gradually decreases, so that the model can keep sensitivity for an abnormal input. In fact, when 99.9% of the area on χ-square distribution is regarded as normal, the model can detect all the anomalies correctly, since the minimum anomaly score calculated by the model is about 13.34 with $r = 0.9999$. Finally, when r is set to 0.9990, the anomaly score becomes unstable, because the model forgets the past data to predict. As a result, for example, the anomaly score becomes 45.24 at around step 17,500, which is larger than that of the truly anomalous data at around step 12,500. We cannot use the same threshold as with $r = 0.9999$ because

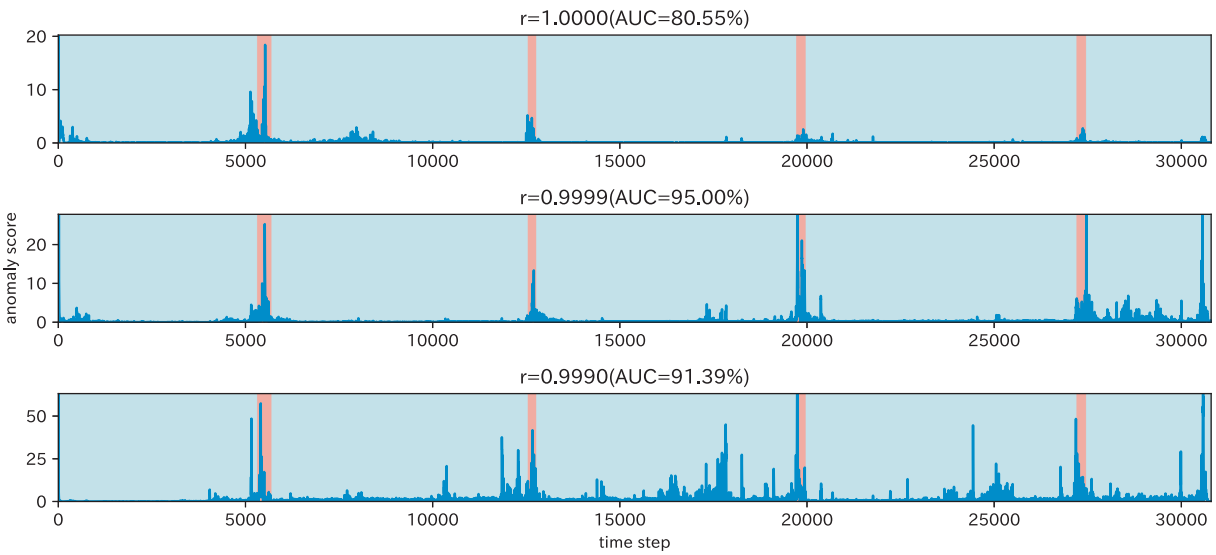


Fig. 6 Relationship between anomaly score and time step for three forgetting rates ($r = 1.0000$, $r = 0.9999$, and $r = 0.9990$)

Table 5 AUC scores for RLS-ESN with different c parameters

c	AUC [%]
0.4	89.91
0.2	89.98
0.1	93.70
0.05	95.00
0.025	94.24
0.0125	94.44

Table 6 AUC scores for RLS-ESN with different δ parameters

δ	AUC [%]
0.6	91.24
0.5	95.00
0.4	91.97

Table 7 Post-synthesis logic area of each model

Model	Area [μm^2]	mem0 size [byte]
OS-ELM core w/o scoring	812,900	10,290
RLS-ESN core w/o scoring	820,648	13,720
RLS-ESN core w/ scoring	822,779	13,720

the anomaly scores with $r = 0.9990$ are generally higher than those with $r = 0.9999$. Still, the score sensitively reacts against anomalous input and it can work more stably than the case of $r = 1.0000$.

6.7 Other Parameters vs. Accuracy

In this section, we show the AUC scores for RLS-ESN with different c and δ parameters. Table 5 shows the AUC scores for RLS-ESN with different c parameters. There is a relationship between c and the AUC score, but it is small. Even if c is double or half the value of 0.05, the effect on the AUC score is about 1%. The relationship between δ and AUC score is shown in Table 6. The AUC score is highest when δ is set to 0.5, but the RLS-ESN achieves relatively high AUC scores of over 90% with a range of $0.4 \leq \delta \leq 0.6$. This means that changes of δ act more robustly on AUC score than those of forgetting rate for $\boldsymbol{\beta}$. RLS-ESN has more parameters than OS-ELM due to the complexity of the structure, but these parameters can be successfully determined as mentioned above.

6.8 Hardware Amount

Table 7 shows the comparison of the existing OS-ELM core [2] and the RLS-ESN core in terms of logic area after the design synthesis. Hardware overhead for the RLS-ESN core is small. The RLS-ESN core is about 1.25 times larger than the OS-ELM core due to the additional registers to store the hidden-layer output which are fed back to the input layer. The size of *mem*0 in Fig. 3 also increases for the same reason. The overhead for the scoring module is actually small. When we include the scoring module, the post-synthesis logic area increases by no more than 1%.

6.9 Execution Latency

Figure 7 shows the prediction, training, and scoring latencies of each implementation. OS-ELM core and RLS-ESN core are the hardware implementations. Their software implementations are executed on the embedded CPU listed in Table 1. As for prediction latency, the RLS-ESN core is about 1.64 times slower than the OS-ELM core because of the increase in input dimensions associated with feedback connections from the previous hidden layer to the input data. On the other hand, the training latency is the same in both algorithms since the matrix $\boldsymbol{\beta}_i$ has the same size. As a result, the overall latency of the RLS-ESN core is not much slower than that of the OS-ELM core. In the RLS-ESN core, the latency overhead for the scoring is small compared to the total execution time (i.e., less than 1%). The hardware implementation of RLS-ESN is about 2.86 times faster than the software implementation (running at 650 MHz) when the RLS-ESN core is implemented as a small dedicated circuit running at 100 MHz.

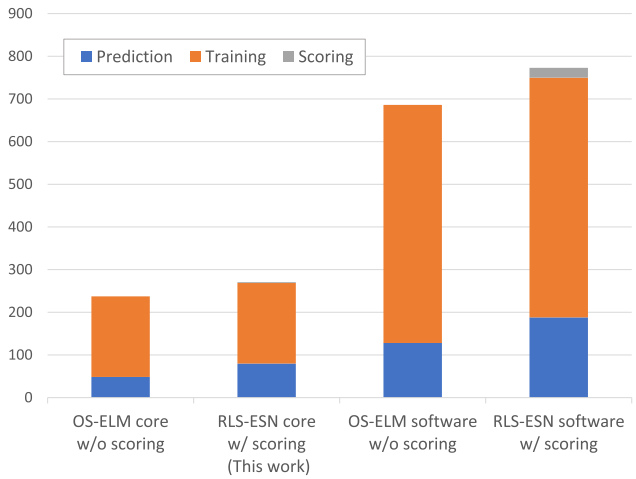


Fig. 7 Latencies for prediction, training, and scoring of each implementation

Table 8 Training time of each model

Model	Training time [s]
LSTM-AD	30484.809
OS-ELM	85.155238
RLS-ESN	93.099750
OS-ELM core	29.454278
RLS-ESN core	33.349572

Table 8 shows the training time of each model with data from subjects 1 and 10-12. OS-ELM and RLS-ESN are trained with RLS, and LSTM-AD is trained with SGD. The training time of LSTM-AD is longer than that of RLS-ESN because of iterative processing by SGD and a large number of parameters. Since OS-ELM and RLS-ESN are trained with RLS, they take a shorter training time than LSTM-AD as shown in Fig. 7.

7. Conclusions

Toward on-device anomaly detection for time-series data in the real environment, this paper proposed an anomaly detection method based on RLS-ESN as a simple form of RNNs.

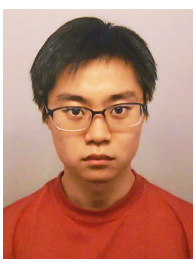
We showed that, the proposed RLS-ESN core can obtain a high anomaly detection capability comparable to LSTM-AD despite its lightness. The additional hardware and latency overheads for the RLS-ESN core are reasonable compared to the existing OS-ELM core [2] without feedback structure.

In this paper, research target is human behavior anomaly detection. Although the proposed RLS-ESN core achieves better anomaly detection capability for the HAR dataset, further investigation is required when applying it to other task domain. Also, the implementation of the Madgwick filter is out of scope in this paper, but it should be integrated with the RLS-ESN core in practice.

Also, the implementation of the Madgwick filter is out of scope in this paper, but it should be integrated with the RLS-ESN core in practice. If the model proposed in this study is to be deployed as a product, the hardware amount should be kept as small as possible while minimizing the latency. For example, using the characteristics of the weight matrix, the memory is reduced, and it leads to a reduction of the hardware amount. As mentioned in Sect. 6.2, the submatrix $\boldsymbol{\alpha}$ consists of only c or $-c$ in this study. As a future work, by encoding c and $-c$ with 0 and 1, each element of $\boldsymbol{\alpha}$ can be represented in 1 bit.

References

[1] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," Proc. European Symposium on Artificial Neural Networks, pp.89–94, April 2015.
[2] 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.
[3] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," ACM Comput. Surv., vol.46, no.4, March 2014.
[4] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks," German National Research Center for Information Technology, GMD Report, vol.148, Jan. 2001.
[5] T. Sakuma and H. Matsutani, "An area-efficient implementation of recurrent neural network core for unsupervised anomaly detection," Proc. IEEE Symposium in Low-Power and High-Speed Chips and Systems (COOL CHIPS), pp.1–3, April 2020.
[6] H. Hotelling, "The generalization of student's ratio," Ann. Math. Statist., vol.2, no.3, pp.360–378, Aug. 1931.
[7] M.I. Jordan, "Chapter 25 - serial order: A parallel distributed processing approach," in Neural-Network Models of Cognition, ed. J.W. Donahoe and V.P. Dorsel, Advances in Psychology, vol.121, pp.471–495, North-Holland, 1997.
[8] J.L. Elman, "Finding structure in time," Cognitive Science, vol.14, no.2, pp.179–211, March 1990.
[9] M. Arnold, R.K.E. Bellamy, M. Hind, S. Houde, S. Mehta, A. Mojsilović, R. Nair, K.N. Ramamurthy, A. Olteanu, D. Piorkowski, D. Reimer, J. Richards, J. Tsay, and K.R. Varshney, "FactSheets: Increasing trust in AI services through supplier's declarations of conformity," IBM J. Research and Development, vol.63, no.4/5, pp.6:1–6:13, July-Sept. 2019.
[10] B. Farhang-Boroujeny, Adaptive Filters: Theory and Applications, Wiley, 1998.
[11] H. Jaeger, "Adaptive nonlinear system identification with echo state networks," Proc. Int. Conf. Neural Information Processing Systems, pp.609–616, Jan. 2002.
[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol.9, no.8, pp.1735–1780, Nov. 1997.
[13] P.R. Vlachas, J. Pathak, B.R. Hunt, T.P. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos, "Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics," Neural Networks, vol.126, pp.191–217, June 2020.
[14] N. Liang, G. Huang, P. Saratchandran, and N. Sundararajan, "A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks," IEEE Trans. Neural Networks, vol.17, no.6, pp.1411–1423, Nov. 2006.
[15] D. Sussillo and L.F. Abbott, "Generating coherent patterns of activity from chaotic neural networks," Neuron, vol.63, no.4, pp.544–557, Aug. 2009.
[16] M.S. Kulkarni and C. Teuscher, "Memristor-based reservoir computing," Proc. IEEE/ACM Int. Symp. Nanoscale Architectures (NANOARCH), pp.226–232, July 2012.
[17] M.L. Alomar, V. Canals, V. Martínez-Moll, and J.L. Rosselló, "Low-cost hardware implementation of reservoir computers," Proc. 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), pp.1–5, Sept. 2014.
[18] Y. Yi, Y. Liao, B. Wang, X. Fu, F. Shen, H. Hou, and L. Liu, "FPGA based spike-time dependent encoder and reservoir design in neuromorphic computing processors," Microprocessors and Microsystems, vol.46, pp.175–183, Oct. 2016.
[19] H. Jaeger, "Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach," German National Research Center for Information Technology, GMD Report, vol.159, Oct. 2002.
[20] M. Munoz-Organero, "Outlier Detection in Wearable Sensor Data for Human Activity Recognition (HAR) Based on DRNNs," IEEE Access, vol.7, pp.74422–74436, May. 2019.
[21] G. Cooper, I. Sheret, L. McMillan, L. McMillian, K. Siliverdis, N. Sha, D. Hodgins, L. Kenney, and D. Howard, "Inertial sensor-based knee flexion/extension angle estimation," J. Biomechanics, vol.42, no.16, pp.2678–2685, Dec. 2009.
[22] S.O.H. Madgwick, "An efficient orientation filter for inertial and inertial/magnetic sensor arrays," Technical report, University of Bristol, UK, 2010.



Takuya Sakuma received the BE degree from Keio University in 2020. He is currently a master course student in Keio University.

Hiroki Matsutani received the BA, ME, and Ph.D. degrees from Keio University 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 and interconnection networks.