An Area-Efficient Implementation of Recurrent Neural Network Core for Unsupervised Anomaly Detection

Takuya Sakuma and Hiroki Matsutani
Dept. of Information and Computer Science, Keio University, JAPAN
E-mail: {sakuma,matutani}@arc.ics.keio.ac.jp

Abstract Toward on-device anomaly detection for time-series data, in this paper, we analyze Echo State Network (ESN), which is a simple form of Recurrent Neural Networks (RNNs), and propose its area-efficient implementation. It is evaluated in terms of the anomaly detection capability and area. (Keywords: On-device learning, Machine learning, and Anomaly detection)

1 Introduction

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 RNNs with a feedback structure, such as Long Short Term Memory cells (LSTMs). However, their training phase based on stochastic gradient descent (SGD) is computationally expensive for edge devices. This issue is addressed by executing their training on high-performance server machines, but it introduces a communication overhead and additional power consumption. On the other hand, ESNs are simple RNNs that can be trained at low cost using a 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 outlier detection as an example of interdependent time-series sensor data. The model is implemented in Verilog HDL, synthesized with a 45nm process technology, and evaluated in terms of the anomaly detection capability and area.

2 Echo State Networks

An ESN is a simple approach to train RNN. As shown in Figure 3, it consists of input layer, hidden layer, and output layer. The numbers of their nodes are n, m, and n', respectively. Given an i-th input $\boldsymbol{x_i} \in \mathbb{R}^{k\times n}$, the model calculates the hidden layer output $\boldsymbol{H}_i$ as follows:

$$\boldsymbol{H_i} \equiv G\Big((1-\delta)\boldsymbol{H_{i-1}} + \delta(\boldsymbol{x_i}\boldsymbol{\alpha} + \boldsymbol{H_{i-1}}\boldsymbol{\gamma})\Big), \tag{1}$$

where $k \in \mathbb{N}$ is a batch size, $\boldsymbol{\alpha} \in \mathbb{R}^{n\times m}$ and $\boldsymbol{\gamma} \in \mathbb{R}^{n\times m}$ are fixed weights, $\delta \in \mathbb{R}$ is a leak rate, and 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 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 an i-th weight connecting the hidden and output layers updated by the least-squares method. For updating β adaptively, we can use the recursive least-squares (RLS) algorithm known as a fast online adaptation method in linear systems. An ESN with the RLS based model update algorithm is known as the recursive least-squares Echo State Network (RLS-ESN) [1].

3 Area-Efficient RNN Core

3.1 Analysis

Equation 1 is heavily used in ESNs. To reduce the number of computation steps for Equation 1 and implement ESNs more compactly, it can be further transformed as follows:

$$\begin{aligned} G\Big((1-\delta)\,\boldsymbol{H_{i-1}} + \delta\,(\boldsymbol{x_i}\boldsymbol{\alpha} + \boldsymbol{H_{i-1}}\boldsymbol{\gamma})\Big) &= G\Big(\boldsymbol{H_{i-1}}\Big((1-\delta)\,\boldsymbol{I} + \delta\boldsymbol{\gamma}\Big) + \boldsymbol{x_i}\delta\boldsymbol{\alpha}\Big) \\ &= G\left(\left[\ \boldsymbol{x_i}\ \middle|\ \boldsymbol{H_{i-1}}\ \right]\left[\frac{\delta\boldsymbol{\alpha}}{(1-\delta)\boldsymbol{I} + \delta\gamma}\right]\right), \end{aligned} \tag{2}$$

where the vertical and horizontal bars denote concatenations of two matrices. This equation shows that, similar to typical neural networks, ESN uses only a simple matrix product to calculate the hidden layer output. The resultant form of the whole algorithm is very close to that of the Online Sequential Extreme Learning Machine (OS-ELM) [2] that also assumes three-layer neural networks. This means that it is possible to implement RLS-ESN as a dedicated circuit in the same way as a simplified hardware design of OS-ELM proposed in [3]. Please note that the RLS-ESN core requires additional registers to hold the previous state since RLS-ESN has a feedback structure as shown in Figure 3. RLS-ESN can apply the same approach of [3] to update the weight β. More specifically, [3] uses the least-squares method with batch size $k = 1$ as follows:

$$\begin{aligned} \boldsymbol{P}_i &= \frac{\boldsymbol{P}_{i-1}}{\lambda} - \frac{\frac{\boldsymbol{P}_{i-1}}{\lambda}\boldsymbol{H}_i^T\boldsymbol{H}_i\frac{\boldsymbol{P}_{i-1}}{\lambda}}{1 + \boldsymbol{H}_i\frac{\boldsymbol{P}_{i-1}}{\lambda}\boldsymbol{H}_i^T} \\ \boldsymbol{\beta}_i &= \boldsymbol{\beta}_{i-1} + \boldsymbol{P}_i\boldsymbol{H}_i^T(\boldsymbol{t}_i - \boldsymbol{H}_i\boldsymbol{\beta}_{i-1}), \end{aligned} \tag{3}$$

where $\boldsymbol{t}_i \in \mathbb{R}^{k\times n'}$ is the correct label. The loss l_i for output $\boldsymbol{y}_i$ is calculated by $l_i = L(\boldsymbol{t}_i, \boldsymbol{y}_i)$, where L denotes a loss function and l_i denotes an anomaly score of $\boldsymbol{x}_i$. In this paper, we use $\boldsymbol{x}_{i+1}$ instead of $\boldsymbol{t}_i$ for the correct label to adopt time-series prediction.

3.2 Anomaly Determination Method

We use Hotelling's T^2 distribution method for anomaly determination. The anomaly score a_i for a given loss l_i is denoted as:

$$a_i = \left(\frac{l_i - \mu}{\sigma} \right)^2, \tag{4}$$

where μ and σ^2 are the mean and variance of loss l_i. A given l_i is detected as anomaly by comparing a_i with a user-defined threshold value. Assuming that the anomaly score follows a χ^2 distribution, the threshold value can be automatically calculated by setting the ratio of outliers to all the observable l_i. The mean and variance usually need to be obtained in advance as fixed parameters in Hotelling's T^2 distribution method. However, since the input distribution is likely to change dynamically in the real world, it is inappropriate to treat the mean and variance as fixed values. Thus, we adopt weighted statistics so that we can follow a concept drift. Algorithm 1 shows the way to calculate weighted statistics adaptively. Here, r is the forgetting rate. A smaller r means more likely forget past information.

3.3 Hardware Implementation

Figure 1 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 these operations consist of the following three substeps: 1) update the index; 2) store the computation target in the register; and 3) save the computation result. Figure 2 presents an overview of the hardware implementation of RLS-ESN. The two memories $mem0$ and $mem1$ are implemented in this architecture. $mem0$ holds the weights of each layer, where $\boldsymbol{\zeta}$ is the weight in Equation 2. $mem1$ holds input values to the model. For each matrix operator (e.g. add, mult, and div), one computing unit is implemented. We adopt 32-bit Q20 number as a fixed-point number format.

4 Evaluations

We evaluate the RLS-ESN core in the anomaly detection capability and area. Table 1 shows the comparison targets and parameters. We use these parameters unless otherwise specified. We adopt the Real World (HAR) [4] dataset containing human activity data as an example of sensor data, and regard different actions, such as turning, as abnormal. We synthesize all the designs with Synopsys Design Compiler 2018.06-SP4. The technology library is Nangate 45nm Open Cell Library. The operation frequency is 100 MHz. We use software implementations on ARM Cortex-A9 processor (@650MHz) to compare with hardware implementations. Table 3 shows the Area Under the Curve (AUC) for each model. As shown in the table, RLS-ESN gets high AUC results and comparable to LSTM even if λ is 0.999, whereas OS-ELM is too sensitive to λ and degrades the AUC because it stores all the past information in $\boldsymbol{\beta}$. It means that it is a complicated task for OS-ELM to select an appropriate λ as a hyperparameter. Figure 4 shows the relationship between the degree of anomaly score and the time step for three forgetting rates ($r = 1.0000$, $r = 0.9999$, and $r = 0.999$). In the graph, the cyan range is normal and the red range is abnormal. At $r = 1$, the variance 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, regarding 99.9% of the area on chi-square distribution as normal, the model can detect all the anomalies correctly, since the minimum anomaly score which the model outputted is about 13.34 with $r = 0.999$. Finally, when r is set to 0.999, the anomaly score becomes unstable because the model forgets the past data to predict. Table 2 and 4 compare OS-ELM core and RLS-ESN core in terms of hardware amount and latency. Hardware overhead for the RLS-ESN core is small. RLS-ESN is about 1.25 times larger than the OS-ELM core due to the additional registers to hold the intermediate layer for the previous input. As for prediction and train latency, RLS-ESN is about 1.11 times slower than OS-ELM because of the increase in input dimensions associated with connecting the previous hidden layer to the input data. The hardware implementation of RLS-ESN is about 3.26 times faster than the software implementation though our hardware implementations focus on area reduction.

5 Summary

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 pre-training with a small amount of data, the model can obtain a high anomaly detection capability. The proposed RLS-ESN core was implemented with only 1.25 times larger hardware amount and only 1.11 times longer latency than the optimized OS-ELM core [3]. Our future work is evaluating the RLS-ESN core with various datasets and real sensor data to confirm its anomaly detection capability.

References

[1] Herbert Jaeger. Adaptive nonlinear system identification with echo state networks. *Proc. of the NIPS*, Jun. 2003.

[2] N.Y. Liang, G.B. Huang, P. Saratchandran, and N. Sundararajan. A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks. *IEEE Transactions on Neural Networks*, 17(6):1411–1423, Nov. 2006.

[3] Mineto Tsukada, Masaaki Kondo, and Hiroki Matsutani. A neural network based on-device learning anomaly detector for edge devices. *CoRR*, abs/1907.10147, 2019.

[4] M. Munoz-Organero. Outlier Detection in Wearable Sensor Data for Human Activity Recognition (HAR) Based on DRNNs. *IEEE Access*, 7:74422–74436, May. 2019.

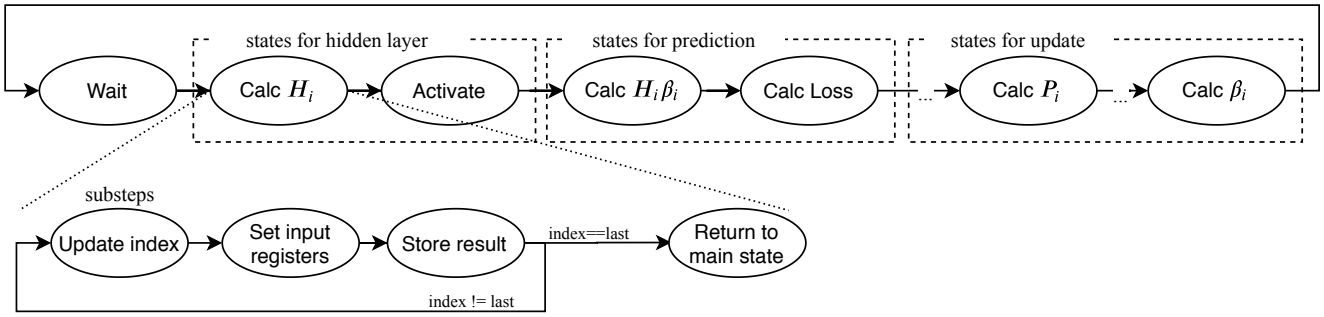

Figure 1: State transition diagram of RLS-ESN

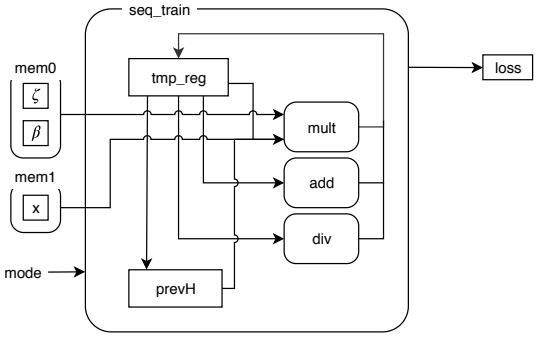

Figure 2: Hardware implementation of RLS-ESN

Algorithm 1 Adaptive calculation of weighted statistics

$r \in (0, 1)$
$A_0 \Leftarrow 0$
$\mu_0 \Leftarrow 0$
$\sigma_0^2 \Leftarrow 0$
for until $\boldsymbol{L_i}$ exists **do**
  $A_i \Leftarrow rA_{i-1} + 1$
  $\mu_i \Leftarrow \frac{rA_{i-1}\mu + L_i}{A_i}$
  $\sigma_i^2 \Leftarrow \frac{rA_{i-1}(\sigma_{i-1}^2 + \mu_i^2) + L_i^2}{A_i} - \mu_i^2$
end for

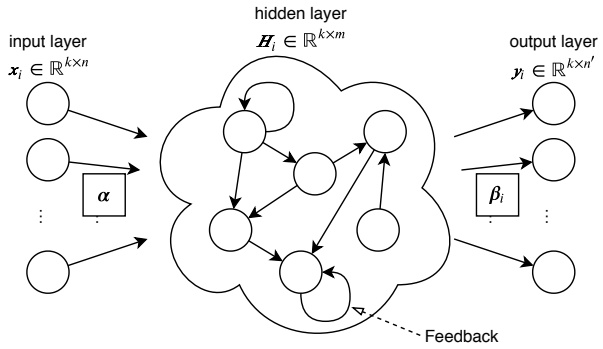

Figure 3: Echo State Network

Table 1: Parameters of each model

	LSTM	OS-ELM	RLS-ESN
Window size	128	1	1
Forgetting rate r for statistics	0.9999	0.9999	0.9999
Number of hidden layer	-	28	28
Forgetting rate λ for weights	-	0.9999	0.9999

Table 2: Hardware amount of each model in hardware implementation

Model	Area [μm^2]	mem0 size [B]	mem1 size [B]
OS-ELM Core	623,229	9,408	112
RLS-ESN Core	775,842	12,544	112

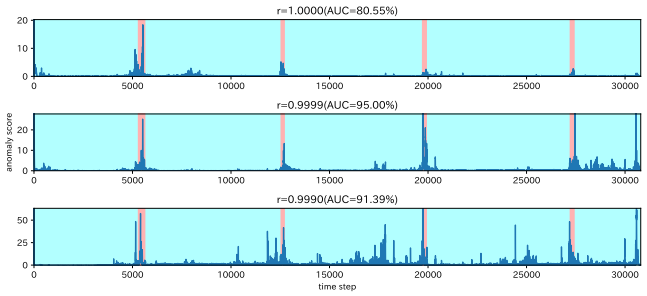

Figure 4: Relationship between degree of anomaly score and time step for three forgetting rates ($r = 1.0000$, $r = 0.9999$, and $r = 0.999$)

Table 3: AUC of each model

Model	AUC [%]
LSTM	94.72
OS-ELM$\lambda = 0.9990$)	65.76
OS-ELM($\lambda = 0.9999$)	94.73
RLS-ESN($\lambda = 0.9990$)	94.55
RLS-ESN($\lambda = 0.9999$)	95.00

Table 4: Latency of each implementation

Model	Prediction [μs]	Prediction & train [μs]
OS-ELM core	4.85	205.93
RLS-ESN core	7.20	229.45
OS-ELM (ARM)	128	686
RLS-ESN (ARM)	188	750