OS-ELM-FPGA: An FPGA-Based Online Sequential Unsupervised Anomaly Detector

Mineto Tsukada[1(✉)], Masaaki Kondo[2], and Hiroki Matsutani[1]

[1] Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan
{tsukada,matutani}@arc.ics.keio.ac.jp

[2] The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan
kondo@hal.ipc.i.u-tokyo.ac.jp

Abstract. Autoencoder, a neural-network based dimensionality reduction algorithm has demonstrated its effectiveness in anomaly detection. It can detect whether an input sample is normal or abnormal by just training only with normal data. In general, Autoencoder is built on backpropagation-based neural networks (BP-NNs). When BP-NNs are implemented in edge devices, they are typically specialized only for prediction with weight matrices precomputed offline due to the high computational cost. However, such devices cannot be immediately adapted to time-series trend changes of input data. In this paper, we propose an FPGA-based unsupervised anomaly detector, called OS-ELM-FPGA, that combines Autoencoder and an online sequential learning algorithm OS-ELM. Based on our theoretical analysis of the algorithm, the proposed OS-ELM-FPGA completely eliminates matrix pseudoinversions while improving the learning throughput. Simulation results using open-source datasets show that OS-ELM-FPGA achieves favorable anomaly detection accuracy compared to CPU and GPU implementations of BP-NNs. Learning throughput of OS-ELM-FPGA is 3.47x to 27.99x and 5.22x to 78.06x higher than those of CPU and GPU implementations of OS-ELM. It is also 3.62x to 36.15x and 1.53x to 43.44x higher than those of CPU and GPU implementations of BP-NNs.

1 Introduction

Autoencoder, a neural-network-based dimensionality reduction algorithm has demonstrated its effectiveness in anomaly detection [3,4,15,17]. Autoencoder constrains the number of hidden nodes to be less than those of input and output nodes, and is trained so that it reconstructs input data in its output. When the reconstruction error between the input and output data is converged well, the dimensionality reduction is completed in the hidden nodes. Since the model uses input data as target data, we can train it in a unsupervised manner.

In a context of anomaly detection, the model is trained using only normal data. When input data that have different patterns from the normal data are

https://doi.org/10.1007/978-3-030-10549-5_41

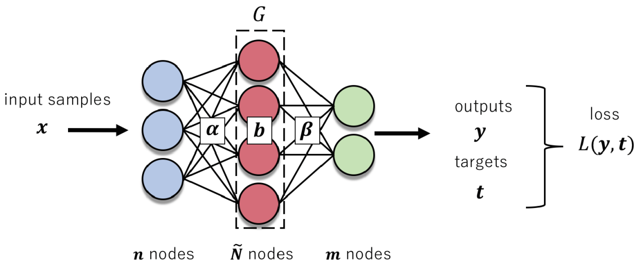

Fig. 1. Single Hidden Layer Feedforward Network (SLFN)

fed to the model, the reconstruction error will increase. If the error exceeds a threshold, the input data can be considered as abnormal data.

In general, Autoencoder is usually built on backpropagation-based neural networks (BP-NNs), and their training is accelerated with GPU-based massively parallel batch processing. For this reason, when BP-NNs are implemented in edge devices, they are typically specialized only for prediction with weight matrices precomputed offline. However, such prediction-only systems cannot immediately follow trend changes of input data. Thus, an anomaly detector that can train online is a primary solution for practical problems where input data trend or noise pattern shift dynamically as time goes by.

In this paper, by making use of Autoencoder and an online sequential learning algorithm OS-ELM, we propose an FPGA-based unsupervised anomaly detector, called OS-ELM-FPGA. OS-ELM [13] is one of neural-network-based convex optimization models. It can train faster than BP-NNs and always find the global optimal solution for its weight matrices at each training. Our theoretical analysis of the algorithm demonstrates the proposed OS-ELM-FPGA completely eliminates costly matrix inversions while improving the learning throughput by fixing the batch size to one.

The rest of this paper is organized as follows. Section 2 briefly introduces OS-ELM and anomaly detection using Autoencoder as background for OS-ELM-FPGA. Section 3 introduces related work. Section 4 proposes our OS-ELM-FPGA and Sect. 5 illustrates the implementation. Section 6 evaluates it in terms of learning throughput, prediction throughput, anomaly detection accuracy, and resource utilization. Section 7 summarizes this paper.

2 Preliminaries

2.1 ELM and OS-ELM

Before introducing OS-ELM, we briefly introduce ELM (Extreme Learning Machine) [9] as background.

ELM is one of single hidden layer feedforward neural networks (SLFNs) illustrated in Fig. 1. In an SLFN, m-dimensional k outputs $\boldsymbol{y} \in \boldsymbol{R}^{k\times m}$ corresponding to n-dimensional k input samples $\boldsymbol{x} \in \boldsymbol{R}^{k\times n}$ are computed by $\boldsymbol{y} = G(\boldsymbol{x}\cdot\boldsymbol{\alpha}+\boldsymbol{b})\boldsymbol{\beta}$, where $\boldsymbol{\alpha} \in \boldsymbol{R}^{n \times \tilde{N}}$ and $\boldsymbol{b} \in \boldsymbol{R}^{\tilde{N}}$ are parameters of the hidden layer. The former is the weight matrix connecting the input layer and the hidden layer, while the latter is the bias vector of the hidden nodes. $\boldsymbol{\beta} \in \boldsymbol{R}^{\tilde{N} \times m}$ is a weight matrix connecting the hidden layer and the output layer, and G is an activation function applied to the hidden nodes.

If the SLFN with $\tilde{N}$ hidden nodes can approximate m-dimensional k targets $\boldsymbol{t} \in \boldsymbol{R}^{k \times m}$ with zero error, it implies that there exists $\boldsymbol{\beta}$ that satisfies the following equation.

$$G(\boldsymbol{x} \cdot \boldsymbol{\alpha} + \boldsymbol{b})\boldsymbol{\beta} = \boldsymbol{t} \tag{1}$$

Then, if we define $\boldsymbol{H} \equiv G(\boldsymbol{x} \cdot \boldsymbol{\alpha} + \boldsymbol{b}) \in \boldsymbol{R}^{k \times \tilde{N}}$, the optimized weight matrix $\hat{\boldsymbol{\beta}}$ is calculated as follows.

$$\hat{\boldsymbol{\beta}} = \boldsymbol{H}^{\dagger}\boldsymbol{t}, \tag{2}$$

where $\boldsymbol{H}^{\dagger}$ is a pseudoinverse of $\boldsymbol{H}$. It can be computed with SVD (Singular Value Decomposition). By just updating the initial $\boldsymbol{\beta}$ with $\hat{\boldsymbol{\beta}}$, the training phase completes. The weight matrix $\boldsymbol{\alpha}$ does not have to be updated once initialized with random values. Furthermore, $\hat{\boldsymbol{\beta}}$ is always the global optimal solution, while BP-NNs is required to address the local minima problem [8]. Please note that ELM assumes all the training samples are available at the training phase in advance.

OS-ELM (Online Sequential Extreme Learning Machine) [13] is an ELM-based algorithm extended to learn input samples one-by-one or chunk-by-chunk.

Given the ith chunk of k_i training samples $\{\boldsymbol{x_i} \in \boldsymbol{R}^{k_i \times n}, \boldsymbol{t_i} \in \boldsymbol{R}^{k_i \times m}\}$, we have to find the optimized $\boldsymbol{\beta_i}$ that minimizes the following prediction error.

$$\left\| \begin{bmatrix} \boldsymbol{H_0} \\ \vdots \\ \boldsymbol{H_i} \end{bmatrix} \boldsymbol{\beta_i} - \begin{bmatrix} \boldsymbol{t_0} \\ \vdots \\ \boldsymbol{t_i} \end{bmatrix} \right\|, \tag{3}$$

where $\boldsymbol{H_i} \equiv G(\boldsymbol{x_i} \cdot \boldsymbol{\alpha} + \boldsymbol{b})$. According to the original paper [13], $\boldsymbol{\beta_i}$ can be sequentially computed with the following equation.

$$\begin{aligned} \boldsymbol{P_i} &= \boldsymbol{P_{i-1}} - \boldsymbol{P_{i-1}}\boldsymbol{H_i^T}(\boldsymbol{I} + \boldsymbol{H_i}\boldsymbol{P_{i-1}}\boldsymbol{H_i^T})^{-1}\boldsymbol{H_i}\boldsymbol{P_{i-1}} \\ \boldsymbol{\beta_i} &= \boldsymbol{\beta_{i-1}} + \boldsymbol{P_i}\boldsymbol{H_i^T}(\boldsymbol{t_i} - \boldsymbol{H_i}\boldsymbol{\beta_{i-1}}) \end{aligned} \tag{4}$$

Specially, $\boldsymbol{P_0}$ and $\boldsymbol{\beta_0}$ can be computed as $\boldsymbol{P_0} = (\boldsymbol{H_0}\boldsymbol{H_0^T})^{-1}$, $\boldsymbol{\beta_0} = \boldsymbol{P_0}\boldsymbol{H_0^T}\boldsymbol{t_0}$. As shown in Eq. 4, $\boldsymbol{\beta_i}$ can be computed without any memories and retraining for the past training samples.

2.2 Anomaly Detection Using Autoencoder

Autoencoder [7] is one of unsupervised learning models that reduces dimensions of input data in its hidden nodes. The model uses input data as target data, and is trained to reconstruct the input data in its output. Since the number of hidden nodes is constrained to be less than those of input and output nodes, when the reconstruction error (e.g., mean squared error and mean absolute error) between the input and the output data is converged well, the dimensionality reduction of the input data is completed in the hidden nodes.

In a context of anomaly detection using Autoencoder, the model is trained only with normal data. When input data that have different patterns from normal data (i.e., abnormal data) are fed, the reconstruction error will increase. If the error exceeds a threshold, the corresponding input data can be considered as abnormal data. Please note that this method does not require any abnormal data or labeling during training. Although PCA (Principal Component Analysis) is often mentioned as a similar model, Sakurada *et al.* showed that Autoencoder can detect subtle anomalies that PCA fails to detect [17] and is easy to apply nonlinear transformation without complex computations that kernel PCA [14] typically requires.

3 Related Work

3.1 Anomaly Detection Using OS-ELM

Since online sequential learning algorithms can follow time-series variability of input data, such algorithms are suitable for anomaly detection where we often have to deal with the nonstationarity. In the past few years, several studies on anomaly detection using OS-ELM have been reported. Singh *et al.* proposed an OS-ELM-based network traffic IDS (Intrusion Detection System) to train fast and accurately on huge amount of network traffic data with a limited memory. Bosman *et al.* presented a decentralized anomaly detection system that can detect abnormality in wireless sensor networks using OS-ELM in an unsupervised manner [11]. Although the above studies apply OS-ELM to anomaly detection, we use OS-ELM in conjunction with Autoencoder. As far as we know, this paper is the first work that uses OS-ELM-based Autoencoder for anomaly detection.

3.2 Hardware Implementation of OS-ELM

Although several hardware implementations of ELM have been reported [16,18,19], there are very few reports on that of OS-ELM. Bosman *et al.* proposed a fixed-point implementation of OS-ELM and its stability correction mechanism for resource-limited embedded devices [10], but they focused on software implementation. In this paper, we implement OS-ELM on an FPGA for the first time and propose an efficient design based on our theoretical analysis discussed in Sect. 4.

4 Analysis on OS-ELM Algorithm

OS-ELM update formula (i.e., Eq. 4) mainly consists of two types of matrix operations: (1) matrix product and (2) matrix inversion. When we assume the number of computational iterations for a matrix product $\boldsymbol{A} \in \boldsymbol{R}^{p \times q} \cdot \boldsymbol{B} \in \boldsymbol{R}^{q \times r}$

is pqr, and that for a matrix inversion $\boldsymbol{C}^{-1} \in \boldsymbol{R}^{r \times r}$ is r^3, the total numbers of iterations for these matrix operations in the update formula are calculated as follows.

$$\begin{aligned} I_{prod} &= 4k\tilde{N}^2 + k(2k + 2m + n)\tilde{N} \\ I_{inv} &= k^3, \end{aligned} \tag{5}$$

where I_{prod} and I_{inv} denote the total numbers of iterations for the matrix products and the matrix inversions, respectively. n, $\tilde{N}$, and m are the numbers of input, hidden, and output nodes of OS-ELM. k is the batch size. For example, we calculated the number of iterations for $\boldsymbol{H_i P_{i-1} H_i^T}$ by dividing it into the following two steps: (1) $\boldsymbol{H_i} \in \boldsymbol{R^{k \times \tilde{N}}} \cdot \boldsymbol{P_{i-1}} \in \boldsymbol{R^{\tilde{N} \times \tilde{N}}}$ and (2) $\boldsymbol{H_i P_{i-1}} \in \boldsymbol{R^{k \times \tilde{N}}} \cdot \boldsymbol{H_i^T} \in \boldsymbol{R^{\tilde{N} \times k}}$, then computing the total number of these iterations $I = k\tilde{N}^2 + k^2\tilde{N}$.

Assuming that I_k denotes the total number of the iterations of matrix products and matrix inversions in OS-ELM update formula when the batch size is k, we can derive the following equation.

$$\begin{aligned} I_k &= I_{prod} + I_{inv} \\ &= 4k\tilde{N}^2 + k(2k + 2m + n)\tilde{N} + k^3 \\ &= k(4\tilde{N}^2 + (2k + 2m + n)\tilde{N} + k^2) \\ &\geq k(4\tilde{N}^2 + (2 + 2m + n)\tilde{N} + 1) = kI_1 \end{aligned} \tag{6}$$

This equation implies that OS-ELM can train at the same or higher learning throughput (i.e., $I_k \geq kI_1$) by fixing the batch size to one. In software frameworks, as actually shown in Sect. 6.3, they suffer from a low throughput at small batch sizes because of software specific overheads, such as dynamic memory allocation and library calls. On the other hand, the proposed FPGA-based implementation of OS-ELM can fully enjoy the insight from Eq. 6, since it is free from any software specific overheads.

Moreover, we can completely eliminate the costly matrix inversions in OS-ELM update formula. Because the size of the target matrix $(\boldsymbol{I} + \boldsymbol{H_i P_{i-1} H_i^T})$ is $k \times k$, its inverse matrix can be easily calculated by computing its reciprocal when $k = 1$. In this case, OS-ELM update formula can be transformed as follows.

$$\begin{aligned} \boldsymbol{P_i} &= \boldsymbol{P_{i-1}} - \frac{\boldsymbol{P_{i-1} h_i^T h_i P_{i-1}}}{1 + \boldsymbol{h_i P_{i-1} h_i^T}} \\ \boldsymbol{\beta_i} &= \boldsymbol{\beta_{i-1}} + \boldsymbol{P_i h_i^T}(\boldsymbol{t_i} - \boldsymbol{h_i \beta_{i-1}}), \end{aligned} \tag{7}$$

where $\boldsymbol{h} \in \boldsymbol{R^{\tilde{N}}}$ is a special case of $\boldsymbol{H} \in \boldsymbol{R^{k \times \tilde{N}}}$ when $k = 1$.

Thanks to the above trick, the proposed OS-ELM-FPGA can train without any costly matrix inversions. It reduces the hardware resources and significantly accelerates the learning throughput. It is possible to further improve the training/prediction throughput by computing matrix products in parallel.

5 Design and Implementation

We implemented the proposed OS-ELM-FPGA using Xilinx Vivado HLS 2016.4 as a toolchain for synthesizing hardware modules written in high-level languages such as C/C++. We chose Xilinx Virtex-7 XC7VX690T as the target FPGA and 100 MHz as the target frequency.

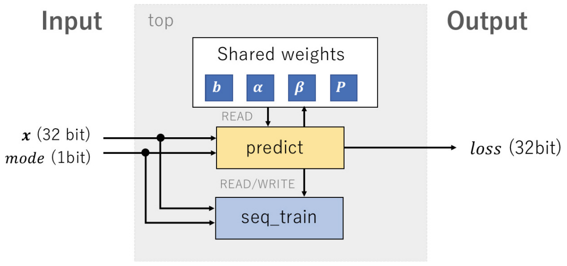

Fig. 2. Block diagram of OS-ELM-FPGA

5.1 Top Module

Figure 2 shows an overview of OS-ELM-FPGA. Since OS-ELM-FPGA uses Autoencoder, the number of input nodes of the network is same as that of output nodes.

top module consists of the following two modules: (1) *seq_train* and (2) *predict* modules. *seq_train* module is to train sequentially on a given input sample $\boldsymbol{x}$, and update shared weight matrices $\boldsymbol{\beta}$ and $\boldsymbol{P}$. *predict* module is to predict a loss (i.e., a reconstruction error) by computing $loss = L(\boldsymbol{x}, \boldsymbol{y})$ where L is a loss function and $\boldsymbol{y}$ denotes an output of the network. Here, we used MAE (Mean Absolute Error) $L(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n}|x_i - y_i|$ as a loss function. Since OS-ELM produces exactly the same learning result regardless of which loss function is used unlike BP-NNs, we recommend to use a loss function that consumes less hardware resources such as MAE.

A 1-bit input signal, named *mode*, determines whether to predict or train on given input samples. When the value is 0 or 1, prediction or training is performed, respectively. In our implementation, all the decimal numbers are represented by 32-bit fixed-point numbers (i.e., 10-bit integer and 22-bit decimal parts).

5.2 Seq_train Module and Predict Module

seq_train module executes OS-ELM-FPGA update formula (i.e., Eq. 7). Figure 3 shows the processing flow of the module. If $1 + \boldsymbol{h_i P_{i-1} h_i^T}$ is close to 0, $\frac{\boldsymbol{P_{i-1} h_i^T h_i P_{i-1}}}{1+\boldsymbol{h_i P_{i-1} h_i^T}}$ will diverge, which makes the training significantly unstable. In our implementation, we set a threshold EPSILON to 1e-4 to detect singular

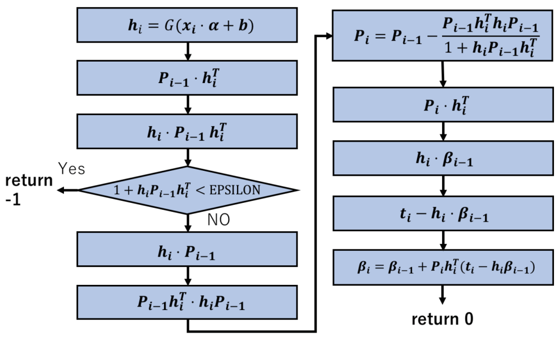
$h_i = G(x_i \cdot \alpha + b)$
$P_{i-1} \cdot h_i^T$
$h_i \cdot P_{i-1} h_i^T$
$1 + h_i P_{i-1} h_i^T <$ EPSILON
Yes
return -1
NO
$h_i \cdot P_{i-1}$
$P_{i-1} h_i^T \cdot h_i P_{i-1}$
$P_i = P_{i-1} - \frac{P_{i-1} h_i^T h_i P_{i-1}}{1 + h_i P_{i-1} h_i^T}$
$P_i \cdot h_i^T$
$h_i \cdot \beta_{i-1}$
$t_i - h_i \cdot \beta_{i-1}$
$\beta_i = \beta_{i-1} + P_i h_i^T (t_i - h_i \beta_{i-1})$
return 0

Fig. 3. Flowchart of Seq_train module

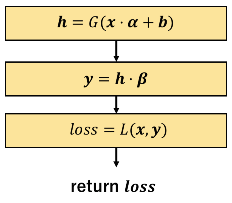

Fig. 4. Flowchart of predict module

matrices. If EPSILON $> 1 + \boldsymbol{h_i P_{i-1} h_i^T}$ is satisfied, OS-ELM-FPGA stops the training and discards the input data for learning stability.

predict module computes output data on given input data by computing matrix/vector products and then the corresponding loss value (i.e., reconstruction error) is computed. Figure 4 shows the processing flow. All the matrix/vector products in *seq_train* module and *predict* module can be accelerated by parallel execution of N product-sums in the innermost loop. This parameter should be tuned by considering the area and performance trade-offs.

6 Evaluations

In this section, OS-ELM-FPGA is evaluated in terms of anomaly detection accuracy, learning throughput, and FPGA resource utilization.

OS-ELM-FPGA is compared with the following four software counterparts: ① OS-ELM(CPU), ② OS-ELM(GPU), ③ BP-NN(CPU), ④ and BP-NN(GPU). ①/② is CPU/GPU implementation of OS-ELM, while ③/④ is CPU/GPU implementation of BP-NN. These counterparts are evaluated on a common server machine (Intel Core i7-6700 (3.4 GHz), NVIDIA GTX 1070 (VRAM 8 GB), DDR4 RAM (32 GB)).

To implement all the four software counterparts, we use Tensorflow [12] (ver 1.6.0). The model size (the numbers of input, hidden, and output nodes) of each implementation is set to 784, 32, and 784 respectively. For OS-ELM-FPGA and OS-ELM(CPU/GPU), we use the linear function $f(\boldsymbol{x}) = \boldsymbol{x}$ as an activation function in their hidden nodes, because it produced better anomaly detection accuracy than other nonlinear functions. In this paper, all the matrix/vector products in OS-ELM-FPGA are fully parallelized in the way mentioned in Sect. 5.2.

For BP-NN(CPU) and BP-NN(GPU), we use relu [20] function in their hidden nodes, and sigmoid [6] function in their output nodes. We use Adam [5] (learning rate = 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$) as the optimization algorithm for them. For all the implementations, we use MAE $L(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n}|x_i - y_i|$

as the loss function as mentioned in Sect. 5.1. AVX (Advanced Vector eXtensions) instructions are used for all the CPU implementations to optimize their performance.

6.1 Anomaly Detection Accuracy

Evaluation Procedure. First, we train each model with a normal training dataset and then perform prediction to compute loss values for a normal validation dataset. Second, we calculate the mean μ and the standard deviation σ of these loss values. Finally, we perform prediction again to calculate another loss values (i.e., *loss*) on a mixed dataset that consists of the normal validation dataset and an abnormal dataset. If $\frac{loss-\mu}{\sigma} > \theta$ is satisfied, the corresponding input sample is detected as abnormal.

Datasets. For the normal dataset, we use MNIST dataset (Fig. 5, 10-class 28×28 gray-scale images) [2]. This dataset consists of 60,000 training samples and 10,000 validation samples. For the abnormal dataset, we use Fashion-MNIST dataset (Fig. 6, 10-class 28×28 gray-scale images) [1]. The dataset also consists of 60,000 training samples and 10,000 validation samples. We use the 10,000 validation samples as abnormal samples. All the samples are fed to OS-ELM-FPGA as $28 \times 28 = 784$-dimensional vector data.

Settings. All the images' pixel values are normalized into [0,1]. The weight matrix $\boldsymbol{\alpha}$ of OS-ELM(CPU/GPU) is initialized with uniform distribution along [0, 1] and then $\boldsymbol{P_0}$ and $\boldsymbol{\beta_0}$ are computed. $\boldsymbol{\alpha}$, $\boldsymbol{P_0}$, and $\boldsymbol{\beta_0}$ of OS-ELM-FPGA are initialized with the same values. Regarding the training procedure, OS-ELM(CPU/GPU) and OS-ELM-FPGA are trained with all the training samples only once (i.e., one epoch), because it makes no sense to train iteratively on the same dataset in OS-ELM algorithm. On the other hand, BP-NN(CPU) and BP-NN(GPU) are trained for ten epochs, because they could not obtain comparable anomaly detection accuracy with one epoch. For all the implementations except for OS-ELM-FPGA, the batch size is set to 64.

Results. Table 1 shows the evaluation results of OS-ELM-FPGA and the four counterparts in terms of precision, recall, and f-measure. In this paper, precision (denoted by P) means a percentage of actual abnormal samples to all the samples detected as abnormal, while recall (denoted by R) is a percentage of samples detected as abnormal to all the actual abnormal samples. F-measure (denoted

Fig. 5. MNIST

Fig. 6. Fashion-MNIST

Table 1. Anomaly detection accuracy

Implementation	θ	P	R	F
OS-ELM-FPGA	1.0	0.852	0.922	0.886
OS-ELM(CPU)	1.0	**0.858**	**0.926**	**0.891**
OS-ELM(GPU)	1.0	0.856	0.924	0.889
BP-NN(CPU)	1.0	0.852	0.908	0.879
BP-NN(GPU)	1.0	0.851	0.901	0.875
OS-ELM-FPGA	3.0	0.996	**0.770**	**0.868**
OS-ELM(CPU)	3.0	0.996	0.765	0.865
OS-ELM(GPU)	3.0	**0.996**	0.747	0.854
BP-NN(CPU)	3.0	0.991	0.662	0.794
BP-NN(GPU)	3.0	0.992	0.669	0.799

Table 2. FPGA resource utilization of OS-ELM-FPGA

	BRAM	DSP	FF	LUT
Used	816	3,347	182,825	330,881
Available	2,940	3,600	866,400	433,200
Utilization	27%	92%	21%	76%

by F) means a harmonic mean of precision and recall. Here, we call f-measure as "anomaly detection accuracy".

Anomaly detection accuracy of OS-ELM-FPGA is higher than those of BP-NN(CPU) and BP-NN(GPU) by up to 9.74%. Considering that they are trained for ten epochs while OS-ELM-FPGA is once, we can say OS-ELM-FPGA achieved better anomaly detection accuracy in a short training time. The result of OS-ELM-FPGA is slightly different from the other OS-ELM counterparts, because we use 32-bit fixed-point to handle the numerical values instead of 32-bit floating-point.

Please note that fixing batch size to one does not affect the accuracy, because OS-ELM always produce the same training result regardless of the batch size.

6.2 FPGA Resource Utilization

OS-ELM-FPGA is evaluated in terms of FPGA resource utilization. Table 2 shows the result. As described in Sect. 5, the target FPGA is Xilinx Virtex-7 XC7VX690T. As shown in the table, all the resource utilizations are less than their limit, though we fully parallelized all the matrix/vector products in OS-ELM-FPGA. The target FPGA device is not the state-of-the-art FPGA already and we can expect faster training/prediction throughput by using the latest FPGAs.

6.3 Sequential Learning Throughput

OS-ELM-FPGA is compared with the counterparts in terms of learning throughput by varying the batch size. Figure 7 shows the result. Since the batch size of OS-ELM-FPGA is one, its learning throughput is constant regardless of x-axis (batch size) in the graph. As shown in Fig. 7, although OS-ELM(CPU) can train faster than BP-NN(CPU) at small batch sizes, the tendency is inverted at big batch sizes since the computational cost for a matrix inversion in OS-ELM

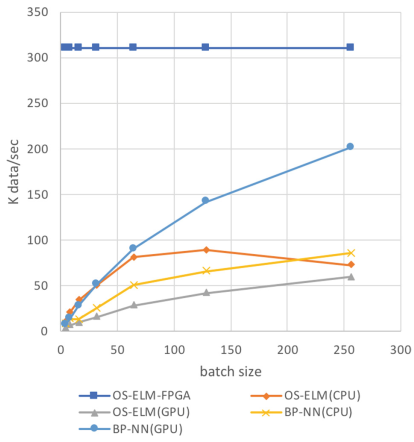

Fig. 7. Comparison of learning throughput

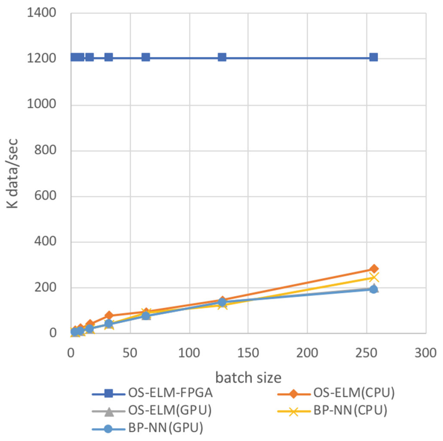

Fig. 8. Comparison of prediction throughput

update formula is proportional to the cube of the batch size. In the context of realtime anomaly detection, it is required to detect abnormal samples immediately after the samples are fed to the detector, thus small batch sizes are preferred in this case. This is a benefit of OS-ELM compared to BP-NNs. In addition, since OS-ELM-FPGA eliminates the computational bottleneck of OS-ELM by fixing the batch size to one, and computes matrix products in parallel, its learning throughput is 3.47x to 27.99x higher than OS-ELM(CPU), and 5.22x to 78.06x higher than OS-ELM(GPU), respectively. It is 3.62x to 36.15x and 1.53x to 43.44x higher than BP-NN(CPU) and BP-NN(GPU), respectively.

Regarding the GPU implementations, while BP-NN(GPU) significantly accelerates its learning throughput, OS-ELM(GPU) suffers from a lower throughput than OS-ELM(CPU). Since a matrix inversion in OS-ELM algorithm is difficult to execute in parallel because of a number of conditional operations, OS-ELM(GPU) could not accelerate the learning throughput efficiently. This result indicates that OS-ELM is less suitable for GPU acceleration than BP-NNs.

On the other hand, since the proposed OS-ELM-FPGA completely eliminates the matrix inversion, it achieves the best learning throughput among all the counterparts for all the batch sizes.

6.4 Prediction Throughput

OS-ELM-FPGA is compared with the counterparts in terms of prediction throughput by varying the batch size. Figure 8 shows the result. Regarding the CPU implementations, OS-ELM(CPU) achieved a slightly higher prediction throughput than that of BP-NN(CPU), because BP-NN(CPU) uses nonlinear activation functions while OS-ELM(CPU) does not use them.

Regarding the GPU implementations, although they execute prediction computations (e.g., matrix products, and matrix sums) in parallel, their throughput decreases on the contrary. When a model size is small like SLFNs, data transfer overheads between a host and GPU devices become major bottlenecks. For this reason, OS-ELM(GPU) and BP-NN(GPU) failed to accelerate the prediction speed. On the other hand, OS-ELM-FPGA is completely free from the data transfer overheads and achieves 4.23x to 83.98x and 6.04x to 183.79x higher throughput than OS-ELM(CPU) and OS-ELM(GPU), respectively. It is also 4.90x to 198.85x and 6.25x to 213.06x higher than NN-GPU(CPU) and NN-BP(GPU), respectively.

7 Summary

In this paper, we proposed an FPGA-based unsupervised anomaly detector, called OS-ELM-FPGA, that combines Autoencoder and an online sequential learning algorithm OS-ELM. Our theoretical analysis demonstrated that the design of OS-ELM-FPGA completely eliminates matrix pseudoinversions while improving the learning throughput. As a result, OS-ELM-FPGA can train and predict using only basic matrix operations, such as matrix product, addition, and subtraction. Simulation results using a hand-written digits dataset and a fashion items dataset showed that OS-ELM-FPGA achieved favorable anomaly detection accuracy compared to CPU and GPU implementations of BP-NN in a short training time. Learning throughput of OS-ELM-FPGA is 3.47x to 27.99x and 5.22x to 78.06x higher than CPU and GPU implementations of OS-ELM, while 3.62x to 36.15x and 1.53x to 43.44x higher than CPU and GPU implementations of BP-NN.

Please note that this paper is the first work that combines OS-ELM and Autoencoder and eliminates the matrix inversions for the efficient FPGA-based online sequential learning unsupervised anomaly detector. In anomaly detection for industries, because environmental noise differs by place and time, our online sequential unsupervised approach is preferable since it can adapt to a given environment online. As future work, we will extend this work to use multiple OS-ELM-FPGA instances in an ensemble manner to improve the expression capability. We will also conduct comprehensive comparisons between OS-ELM-FPGA and some other methods (e.g., PCA and kernel PCA) on more practical scenario using real industrial data.

Acknowlegements. This work was supported by JST CREST Grant Number JPMJCR1785, Japan.

References

1. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. https://github.com/zalandoresearch/fashion-mnist
2. MNIST: Handwritten digit database. http://yann.lecun.com/exdb/mnist/
3. Zhou, C., Paffenroth, C.: Anomaly detection with robust deep autoencoders. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 665–674, August 2017

4. Chicco, D., Sadowski, P., Baldi, P.: Deep autoencoder neural networks for gene ontology annotation predictions. In: Proceedings of the ACM Conference on Bioinformatics, Computational Biology, and Health Informatics, pp. 533–540, September 2014
5. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. CoRR abs/1412.6980, January 2014
6. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control Signals Syst. **2**(4), 303–314 (1989)
7. Hinton, G., Salakhutdinov, R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006)
8. Marco, G., Alberto, T.: On the problem of local minima in backpropagation. IEEE Trans. Pattern Anal. Mach. Intell. **14**(1), 76–86 (1992)
9. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: a new learning scheme of feedforward neural networks. In: Proceedings of the International Joint Conference on Neural Networks, pp. 985–990, July 2004
10. Bosman, H.H.W.J., Liotta, A., Iacca, G., Wörtche, H.J.: Online extreme learning on fixed-point sensor networks. In: Proceedings of the IEEE International Conference on Data Mining Workshops, pp. 319–326, December 2013
11. Bosman, H.H.W.J., Iacca, G., Tejada, A., Wörtche, H.J., Liotta, A.: Spatial anomaly detection in sensor networks using neighborhood information. Inf. Fusion **33**, 41–56 (2017)
12. Abadi, M., et al.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, March 2016. https://www.tensorflow.org/
13. Liang, N.Y., Huang, G.B., Saratchandran, P., Sundararajan, N.: A fast and accurate online sequential learning algorithm for feedforward networks. IEEE Trans. Neural Netw. **17**(6), 1411–1423 (2006)
14. Wang, Q., et al.: Kernel Principal Component Analysis. In: Artificial Neural Networks. pp. 583–588, July 1997
15. Fakoor, R., Ladhak, F., Nazi, A., Huber, M.: Using deep learning to enhance cancer diagnosis and classification. In: Proceedings of the International Conference on Machine Learning, vol. 28, August 2013
16. Decherchi, S., Gastaldo, P., Leoncini, A., Zunino, R.: Efficient digital implementation of extreme learning machines for classification. IEEE Trans. Circ. Syst. II: Express Briefs **59**(8), 496–500 (2012)
17. Mayu, S., Takehisa, Y.: Anomaly detection using autoencoders with nonlinear dimensionality reduction. In: Proceedings of the Workshop on Machine Learning for Sensory Data Analysis, pp. 4–11, July 2014
18. Yeam, T.C., Ismail, N., Mashiko, K., Matsuzaki, T.: FPGA implementation of extreme learning machine system for classification. In: Proceedings of the IEEE Region 10 Conference, pp. 1868–1873, November 2017
19. Frances, V., et al.: Hardware implementation of real-time extreme learning machine in FPGA: analysis of precision, resource occupation and performance. Comput. Electr. Eng. **51**, 139–156 (2016)
20. Nair, V., Hinton, G.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the International Conference on Machine Learning, pp. 807–814, June 2010