Performance and Cost Evaluations of Online Sequential Learning and Unsupervised Anomaly Detection Core

Tomoya Itsubo, Mineto Tsukada, and Hiroki Matsutani
Dept. of Information and Computer Science, Keio University, JAPAN
E-mail: {itsubo,tsukada,matutani}@arc.ics.keio.ac.jp

Abstract Toward on-device learning on IoT devices, this paper implements an online sequential learning and unsupervised anomaly detection core and explores its design options, such as pipeline structure. They are evaluated in terms of performance and cost. (Keywords: On-device learning, Machine learning and Pipeline structure)

1 Introduction

Anomaly detection is one of practical applications on IoT devices. In practical environments, an expected anomaly detection behavior sometimes varies with time due to noises or environmental changes after the deployment. In addition, collecting training data sets for possible environments beforehand requires time and effort. Therefore, a divergence between expected anomaly detection behavior and training data sets used for building the model is a crucial issue. One approach to address this issue is on-device learning that performs both the anomaly detection and online sequential learning at the same time on edge devices [1]. The on-device learning capability is expected to be embedded to environmental sensor chips. Since their compute resources are strictly limited, in this paper we explore design options of the on-device learning core in terms of pipeline structure and arithmetic unit reuse. The design options are implemented in Verilog HDL, synthesized with a 45nm process technology, and evaluated in terms of on-device learning throughput and area.

2 Related Work

2.1 Online Sequential Extreme Learning Machine (OS-ELM)

OS-ELM [2] is used as an online sequential learning algorithm for single layer feedforward networks that consist of input layer, hidden layer, and output layer. The numbers of their nodes are n, m, and n'. OS-ELM sequentially learns input data $x \in \boldsymbol{R^{k \times n}}$, where k is batch size. As in [1], in this paper k is fixed to one in order to eliminate the pseudo inverse operations of $k \times k$ matrix, except for initial training phase mentioned later. For the i-th input data $x_i \in \boldsymbol{R^{1 \times n}}$, the hidden layer matrix is defined as $h_i \equiv G(x_i \cdot \alpha + b)$ using activation function G, weight matrix $\alpha \in \boldsymbol{R^{n \times m}}$ between the input and hidden layers, and bias $b \in \boldsymbol{R^{m}}$ of the hidden layer. α is initialized with random values. The optimized weight matrix $\beta_i \in \boldsymbol{R^{m \times n'}}$ between the hidden and output layers can be computed by the following equation using intermediate result P_i and training data $t_i \in \boldsymbol{R^{n'}}$.

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

In particular, the initial values P_0 and β_0 are obtained as follows.

$$\begin{aligned} \boldsymbol{P_0} &= (\boldsymbol{H_0} \boldsymbol{H_0^T})^{-1} \\ \boldsymbol{\beta_0} &= \boldsymbol{P_0} \boldsymbol{H_0^T} \boldsymbol{t_0} \end{aligned} \tag{2}$$

Here, H_0 is the initial hidden layer matrix obtained from the initial data $x_0 \in \boldsymbol{R^{k_0 \times n}}$, where k_0 is the initial batch size which should be greater than m. In this paper, P_0 and β_0 computed by software beforehand are fed to the on-device learning core.

2.2 Unsupervised Anomaly Detection

Autoencoder [3] is a dimensionality reduction approach, and it is used for unsupervised anomaly detection in combination with OS-ELM as in [1]. In this case, n and n' are the same. Input data x_i is used also as training data t_i. That is, the weight β_i is trained so that input data x_i is reconstructed by the autoencoder. Assume that an autoencoder has been trained only with normal patterns. In this case, when normal data is fed to the autoencoder, the difference between the input data and reconstructed data (i.e., loss value) is small, while when anomaly data is fed, the loss value becomes large. This behavior is used for the unsupervised anomaly detection.

3 Online Sequential Learning and Unsupervised Anomaly Detection Core

3.1 Overview

Figure 1 illustrates the online sequential learning and unsupervised anomaly detection core [1] written in Verilog HDL. seq_train module is used for the online sequential learning and predict module is used for the anomaly detection. More specifically, seq_train module updates the intermediate result P and weight β from a given input data x using Equation 1. predict module computes $loss$ value that represents the reconstruction error. When the loss value exceeds a certain threshold, the input data is detected as anomaly. In addition to input data, a 1-bit mode signal is fed to the on-device learning core to select either seq_train module (mode = 0) or predict module (mode = 1).

predict module simply computes $|x\alpha\beta - x|$ as $loss$. Both the modules are pipelined. Especially, pipeline structure of seq_train module affects the area and performance. Equation 1 is divided into six stages as shown in Figure 2. A pipeline dependency exists in these stages. Until P of stage4 is updated for input data x_i, stage2 cannot accept the next input data x_{i+1}. Figure 3 (left) illustrates the pipeline processing of data1, data2, and data3 by considering the pipeline dependency. Here, the number of cycles for the j-th stage is denoted as $T_{stage j}$. The number of cycles required for a sequential learning of each input data is represented as $max\{T_{stage1}, T_{stage2} + T_{stage3} + T_{stage4}\}$.

3.2 Pipeline Stage Design

The pipeline structure can be customized by changing the number of arithmetic units. Below are three pipeline designs.

- Design 1 is introduced for the minimum execution cycle. Dedicated arithmetic units are implemented for all the fixed-point matrix operations denoted as fxadd, fxmult, and fxdiv.
- Design 2 is a low cost version, where fxmult units are reused in each stage.
- Design 3 is another low cost version, where fxadd and fxmult units are reused in each stage.

The learning throughput of design1 is the highest, followed by design2 and design3. Using design2 and design3, area for arithmetic units can be reduced. Note that when the same arithmetic unit is reused temporally for different computations, DFFs are required to store temporal results of the arithmetic unit.

In addition, throughputs of all the designs can be improved by speculative learning that performs stage2 of input data x_{i+1} without waiting stage4 of x_i. Figure 3 (right) illustrates the pipeline processing of the speculative learning. In this case, although update of the weight β is slightly delayed, the negative impact is limited because value changes on P and β for each sequential training are quite small. Assume input data trend changes at x_i suddenly and the same trend continues from then. Input data x_i is detected as anomaly. In this situation, the successive inputs may be detected as anomaly too since they will be evaluated with slightly-obsolete P and β. In this speculative pipeline processing, utilization rate of the slowest pipeline stage is 100%. The number of cycles required for a sequential learning is thus shortened to $max\{T_{stage1}, T_{stage4}\}$.

4 Evaluations

All the designs were synthesized with Synopsys Design Compiler I-2013.12-SP2. The technology library used was Nangate 45nm Open Cell Library. Bitwidth of fixed-point arithmetic units (i.e., fxadd, fxmult, and fxdiv) can be customized, and we used 12 bits for integer and 20 bits for fraction. fxadd and fxmult units were implemented as combinational logic, while fxdiv unit was implemented as sequential logic with right-shift and subtract operations. Design space exploration of the arithmetic units is omitted for page limitation.

Tables 1 and 2 show the cost models of seq_train and predict modules for design1, design2, and design3, where S_a, S_m, and S_d are costs of single fxadd, fxmult, and fxdiv units, respectively. Also, n' is equal to n. Figure 4 shows their logic areas [mm^2] when $n = 256$ and $m = 32$ after the logic synthesis. design2 and design3 consume only 26% and 15% of design1, respectively. Figure 5 shows the sequential learning throughputs [M samples per sec] of these designs. Those improved by the speculative pipeline are also shown. By introducing the speculative pipeline, the throughputs of design1, design2, and design3 are improved by 1.1x, 2.2x, and 1.0x, respectively. Note that, in design3, since T_{stage1} is greater than $T_{stage2} + T_{stage3} + T_{stage4}$, the throughput does not change even if the speculative learning is performed. Figure 6 compares the baseline and speculative pipelines in terms of loss values. Input data is generated as $\sin(\theta + \phi)$, where $0^\circ \leq \theta < 360^\circ$ with noise. Those with $\phi = 0^\circ$ are learned as normal. Loss values of the speculative pipeline are close to the baseline.

5 Summary

Toward on-device learning on IoT devices, in this paper the online sequential learning and unsupervised anomaly detection core was implemented with three pipeline structures (i.e., design1, design2, and design3). Speculative learning was also proposed to shorten the execution cycles. They were evaluated in terms of learning throughput and area. The evaluation results showed that design2 plus the speculative learning strikes a good balance between the throughput and area.

We are planning to implement the on-device learning chip. Further design space exploration of arithmetic units and fixed-point number representations in terms of performance, cost, and accuracy is also our future work.

Acknowlegements This work was supported by VDEC, the University of Tokyo in collaboration with Synopsys.

References

[1] Mineto Tsukada, Masaaki Kondo, and Hiroki Matsutani. OS-ELM-FPGA: An FPGA-Based Online Sequential Unsupervised Anomaly Detector. In *Proceedings of the International European Conference on Parallel and Distributed Computing (Euro-Par'18) Workshops*, pages 518–529, Aug 2018.

[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] G. Hinton and R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, Jul 2006.

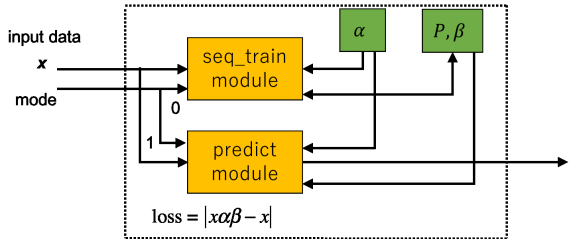

Figure 1: Online sequential learning and unsupervised anomaly detection core

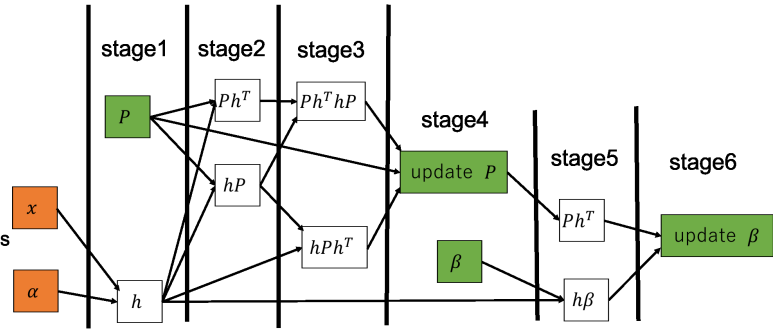

Figure 2: Six pipeline stages of seq_train module

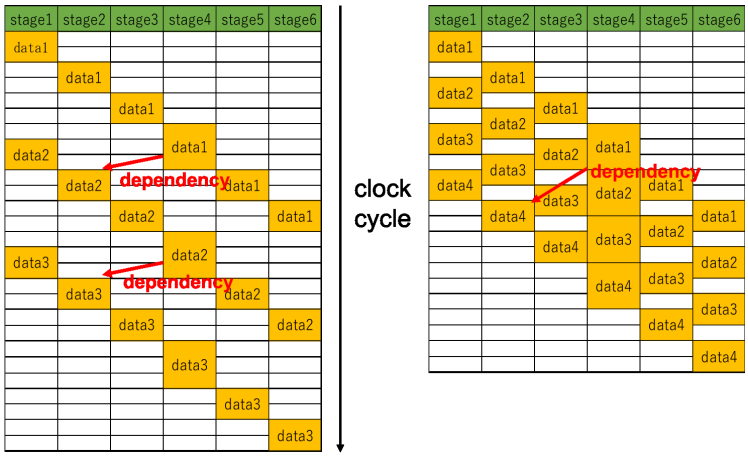

Figure 3: Baseline pipeline (left) and speculative pipeline (right)

Table 1: Cost model of seq_train module

design1	$\left(3nm + 4m^2 - 4m - 2\right) S_a +$ $\left(4m^2 + 3mn + m\right) S_m + m^2 S_d$
design2	$\left(3nm + 4m^2 - 4m - 2\right) S_a +$ $\left(4m + 3n + 1\right) S_m + m^2 S_d$
design3	$\left(m^2 + 2n + 4m + 1\right) S_a +$ $\left(4m + 3n + 1\right) S_m + m^2 S_d$

Table 2: Cost model of predict module

design1	$(2mn + n - m - 1)S_a + 2mnS_m$
design2	$(2mn + n - m - 1)S_a + 2nS_m$
design3	$(2n + m + 1)S_a + 2nS_m$

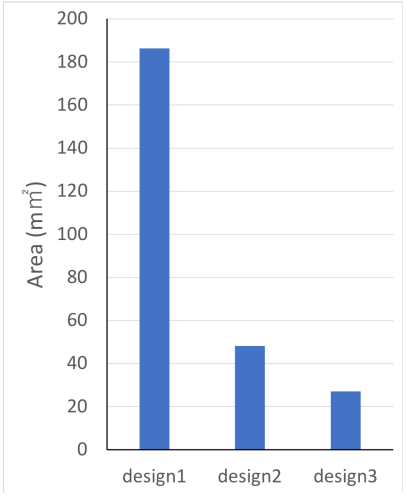

Figure 4: Areas of design1, design2, and design3

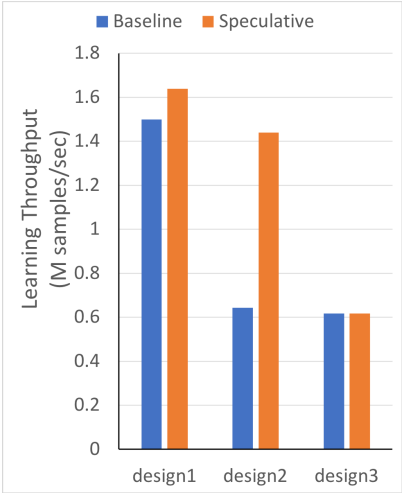

Figure 5: Learning throughputs of design1, design2, and design3

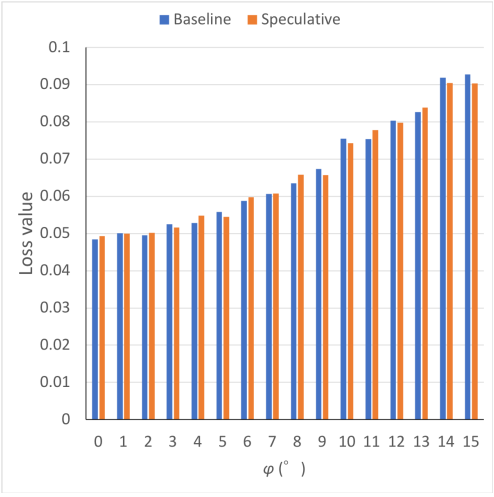

Figure 6: Accuracy of speculative learning