情報処理学会論文誌　コンピューティングシステム　Vol.12　No.3　34–45 (July 2019)

OSUAD：FPGAを用いたオンライン逐次学習型教師なし異常検知器

塚田 峰登[1,a)]　近藤 正章[2,b)]　松谷 宏紀[1,c)]

受付日 2018年12月4日, 採録日 2019年3月26日

概要：教師なし異常検知は，モデルの学習時に異常データを必要としない異常検知手法であり，近年Backpropagation 法を用いたニューラルネットワーク（以降，BP-NN と表記）を用いた手法が利用されている．これらの手法では，通常，あらかじめ用意した正常データを用いて異常検知モデルを学習する．その後，異常検知を行うデバイス群に学習済みモデルを配信し，デバイス側では推論のみを行う．この場合，学習に使用した正常データと実際に与えられる入力データの正常パターンの差異が問題となる．具体的には，(1) BP-NN は学習が収束するまで繰り返し学習をする必要があり，(2) 各デバイスが置かれた環境の変動に対応するには，そのつど，再学習が必要となる．実際の現場では，モデルをそれぞれの環境に特化させることが望ましく，データの特徴は環境や時間とともに変動するため即時的なパラメータ更新が求められている．そこで本研究では，デバイス上で逐次学習ができる FPGA ベースの教師なし異常検知器（OSUAD）を提案する．OSUAD はニューラルネットワーク技術の 1 つであるオートエンコーダと高速な逐次学習アルゴリズムである OS-ELM を組み合わせる．本研究では，OS-ELM の計算量的ボトルネックである逆行列計算を完全に消去することでさらなる高速化と省面積化を達成する．評価の結果，OSUAD はローエンドな FPGA に実装可能なサイズでありながら，CPU と GPU を用いた実装と比較してそれぞれ平均 2.47 倍，4.95 倍高速な学習処理を実現した．また，BP-NN を用いた異常検知モデルと比較して 10 分の 1 の学習エポック数で 2.4%高い F 値を実現した．

キーワード：FPGA，OS-ELM，教師なし異常検知

OSUAD: An FPGA-Based Online Sequential Learning Unsupervised Anomaly Detector

Mineto Tsukada[1,a)]　Masaaki Kondo[2,b)]　Hiroki Matsutani[1,c)]

Received: December 4, 2018, Accepted: March 26, 2019

Abstract: Unsupervised anomaly detection is an anomaly detection approach where abnormal samples are not required to train models, and recently backpropagation-based neural networks (i.e., BP-NNs) have been used. Typically, the models are trained using normal data, and then the pretrained models are installed to devices that perform anomaly detection. The devices themselves perform only inference. In this case, however, difference between the normal data at training stage and actual normal patterns of input data in the deployed environment becomes an issue. More specifically, (1) BP-NNs have to train iteratively until the learning converges, and (2) retraining is required to follow the variation and fluctuation in a deployed environment. In reality, the model should be specialized for a deployed environment, and responsive update of parameters is required to follow the variation and fluctuation. In this paper, we propose an FPGA-based online sequential learning unsupervised anomaly detector, called OSUAD. The detector combines Autoencoder, one of neural-network-based models, and OS-ELM, a fast online sequential learning algorithm. OSUAD achieves faster training and smaller area by completely eliminating costly matrix inversions which are the computational bottlenecks. Evaluation results show that the training latency of OSUAD is faster than CPU and GPU implementations by 2.47x and 4.95x on average, while the detector can be implemented in a low-end FPGA device. It also achieves 2.4% higher f-measure than a BP-NN-based implementation in only 1/10 training epochs.

Keywords: FPGA, OS-ELM, unsupervised anomaly detection

1. はじめに

教師なし異常検知は，モデルの学習時に異常データを必要としない異常検知手法である．あらかじめ与えられた正常データや，小数の異常データが含まれるデータをモデリング，あるいは学習し，それらと異なる特徴を持つ入力データを異常データとして検出する．一般的に異常データをあらかじめ用意するのは困難であるため，学習時に異常データを必要としない点は実用上メリットになる．

教師なし異常検知のアルゴリズムとして Local Outlier Factor [4]，Gaussian Mixture Model [23]，One Class SVM [5] など様々な手法が提案されてきた．しかし，近年は取り扱うデータの次元数が増加しており，次元の呪い [28] によって上記の手法で十分な精度を出すのが困難になった．そのため，現在は，高次元データでも高い精度を実現するニューラルネットワークを用いた教師なし異常検知手法が多数発表されている [8], [24], [25]．中でも，Backpropagation 法ベースのニューラルネットワーク（以降，BP-NN と表記）が広く用いられている．

図 1 の左側に，BP-NN を用いた教師なし異常検知モデルの最も典型的な学習手法を示す．通常，あらかじめ用意した正常データを用いて異常検知モデルを学習し，異常検知を行うデバイス群に学習済みのモデルを配信する．このときデバイス側はパラメータ更新と推論のみを行い，学習は行わない．しかしこの場合，学習に使用した正常データと実際に与えられる入力データの正常パターンの差異が問題となる．具体的には，(1) BP-NN は学習が収束するまで繰り返し学習する必要があり，(2) 各デバイスが置かれた環境の変動に対応するには，そのつど，再学習が必要となる．教師なし異常検知モデルは正常データを学習するが，実世界の正常データの特徴は時間とともに変動する場合があるため，異常検知の精度を維持するには即時的に変動に対応し，パラメータを更新できることが求められる．また，正常データの特徴は環境ごとに異なる場合がある．たとえば，モータの振動データを用いて異常を検出することを想定した場合，どのような振動パターンがそのモータの正常状態にあたるかは，周囲のノイズや他のモータとの位置関係にも依存する．よって，この場合モータごとにモデルを学習するのが望ましいが，図 1 の左側の手法で対応しようとすると環境ごとに 1 つ 1 つモデルを作り分けなければならない．一方で，図 1 の右側に示す手法は上記

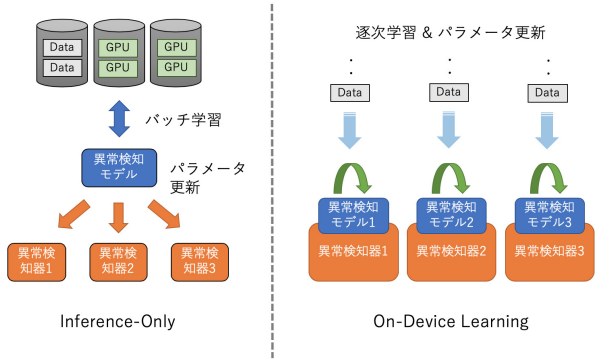


図 1　BP-NN を用いた教師なし異常検知モデルの典型的な学習手法（左図）とデバイス上での逐次学習（右図）

Fig. 1　Typical training method of BP-NN based unsupervised anomaly detection models (left) and on-device sequential training approach (right).

(1) および (2) の問題を回避できる．デバイス上でそれぞれに割り当てられたモデルを逐次的に学習することで正常データの特徴の変動に即時に追従でき，かつデバイスが置かれた環境に特化してモデルを学習できる．しかし，学習に必要な Backpropagation 法は計算コストが大きく，ハードウェア資源の限られたデバイスに実装するのは困難である [20], [21]．現在多くの BP-NN ベースのハードウェアが推論処理に特化しているのはそうした背景がある．

そこで，本研究ではニューラルネットワークの 1 つである OS-ELM（Online Sequential Extreme Learning Machine）[18] に注目する．OS-ELM は近年注目を集める逐次学習アルゴリズムであり，逆行列計算を用いることで BP-NN よりも高速に学習できる [18]．本研究では OS-ELM の計算量的ボトルネックである逆行列計算を完全に消去することで，さらなる高速化と省面積化を実現する．また，教師なし異常検知手法の 1 つであるオートエンコーダを用いた手法 [24] と組み合わせることで，デバイス上で高速に逐次学習可能な FPGA ベースの教師なし異常検知器（Online Sequential learning Unsupervised Anomaly Detector，以降 OSUAD と表記）を提案する[*1]．

本論文の構成は以下のとおりである．1 章で本研究の背景を述べる．2 章では，必要な知識を解説する．3 章で関連研究について述べ，4 章で OSUAD について述べる．5 章では，OSUAD の実装を説明し，6 章で OSUAD の評価を行う．最後に，7 章で本論文をまとめる．

2. 前提知識

2.1 Extreme Learning Machine

OS-ELM の前提知識として，まず ELM（Extreme Learning Machine）[12] について述べる．ELM はニューラルネッ

[1] 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University, Yokohama, Kanagawa 223–8522, Japan
[2] 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
a) tsukada@arc.ics.keio.ac.jp
b) kondo@hal.ipc.i.u-tokyo.ac.jp
c) matutani@arc.ics.keio.ac.jp

[*1] 本論文は，著者らが国際ワークショップ HeteroPar'18 で発表した過去の研究 [27] の評価を拡充したものであり，本論文の提案実装である OSUAD は，文献 [27] における OS-ELM-FPGA と同一である．

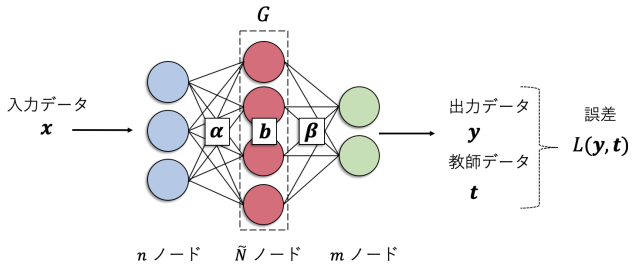


図 2　Extreme Learning Machine
Fig. 2　Extreme Learning Machine.

トワークの 1 つであり，図 **2** にあるように単層ニューラルネットワークの構造をとる．入力層，隠れ層，出力層の三層で構成されるモデルは特に単層ニューラルネットワークと呼ばれる．一般に，単層ニューラルネットワークにおいてバッチサイズ k の n 次元の入力データ $\boldsymbol{x} \in \boldsymbol{R}^{k \times n}$ に対応する m 次元の推論結果 $\boldsymbol{y} \in \boldsymbol{R}^{k \times m}$ は，$\boldsymbol{y} = G(\boldsymbol{x} \cdot \boldsymbol{\alpha} + \boldsymbol{b})\boldsymbol{\beta}$ として得られる．このとき，$\boldsymbol{\alpha} \in \boldsymbol{R}^{n \times \tilde{N}}$ は入力層と隠れ層を結合する重みであり，$\boldsymbol{\beta} \in \boldsymbol{R}^{\tilde{N} \times m}$ は隠れ層と出力層を結合する重みを示す．また，$\boldsymbol{b} \in \boldsymbol{R}^{\tilde{N}}$ と G はそれぞれ隠れ層のバイアスと活性化関数を示す．

ELM の学習手法について述べる．訓練データ $\{\boldsymbol{x} \in \boldsymbol{R}^{k \times n}, \boldsymbol{t} \in \boldsymbol{R}^{k \times m}\}$ が与えられたとする．まず，重み $\boldsymbol{\alpha}$ を任意の乱数で初期化し，もう一方の重み $\boldsymbol{\beta}$ を 0 で初期化する．ここで，入力データ $\boldsymbol{x}$ に対する推論結果 $\boldsymbol{y}$ と教師データ $\boldsymbol{t}$ の誤差が 0 であると仮定すると次の等式が成立する．

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

上式を満たす最適解 $\hat{\boldsymbol{\beta}}$ は隠れ層行列 $\boldsymbol{H} \equiv G(\boldsymbol{x} \cdot \boldsymbol{\alpha} + \boldsymbol{b}) \in \boldsymbol{R}^{k \times \tilde{N}}$ を用いて次の式を計算することで求められる．

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

$\boldsymbol{H}^{\dagger}$ は $\boldsymbol{H}$ の擬似逆行列であり，SVD（Singular Value Decomposition）や QRD（QR Decomposition）などの行列分解アルゴリズムを用いて計算できる．$\boldsymbol{\beta}$ を $\hat{\boldsymbol{\beta}}$ で更新することで学習が完了する．

現在主流の BP-NN との相違点は以下のとおりである．まず BP-NN は $\boldsymbol{\alpha}$ と $\boldsymbol{\beta}$ の両方の最適化を行うが，ELM は $\boldsymbol{\beta}$ のみを最適化し，$\boldsymbol{\alpha}$ は任意の乱数で初期化した後に学習は行わない．また $\hat{\boldsymbol{\beta}}$ は大域最適解であるため，BP-NN の問題点の 1 つである局所最適解への収束 [19] を完全に回避できる．さらに，BP-NN は同じ訓練データを繰り返し学習することで最適化を行うが，ELM は一度の最適化計算で学習が完了するため学習時間全体が大幅に削減される．しかし，ELM はあらかじめすべての学習データが用意されている場合を想定しており，逐次的に訓練データが生成される場合はそのつど過去の訓練データも含めて再学習する必要がある．その場合，過去の全訓練データを保存するための記憶領域が必要になり，最適化計算の計算負荷も増加する．訓練データのバッチサイズを k とすると，計算量は $O(k^3)$ であるため，ELM は逐次的な学習に向かない．

2.2　Online Sequential Extreme Learning Machine

本節では OS-ELM（Online Sequential Extreme Learning Machine）について述べる．OS-ELM [18] は ELM を任意のバッチサイズで逐次的に学習できるように拡張したアルゴリズムである．バッチサイズ k_i の i 番目の訓練データ $\{\boldsymbol{x}_i \in \boldsymbol{R}^{k_i \times n}, \boldsymbol{t}_i \in \boldsymbol{R}^{k_i \times m}\}$ が得られたとすると，次の誤差を最小化する $\boldsymbol{\beta}_i$ を求める必要がある．

$$\left\| \begin{bmatrix} \boldsymbol{H}_0 \\ \vdots \\ \boldsymbol{H}_i \end{bmatrix} \boldsymbol{\beta}_i - \begin{bmatrix} \boldsymbol{t}_0 \\ \vdots \\ \boldsymbol{t}_i \end{bmatrix} \right\| \tag{3}$$

このとき，i 番目の隠れ層行列は $\boldsymbol{H}_i \equiv G(\boldsymbol{x}_i \cdot \boldsymbol{\alpha} + \boldsymbol{b})$ と定義される．また，最適化された重み $\boldsymbol{\beta}_i$ は以下の式で計算できる．

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

特に，$\boldsymbol{P}_0$ と $\boldsymbol{\beta}_0$ については次の式で得られる．

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

ELM では逐次的に学習するには過去の全訓練データを保存する記憶領域が必要であったが，OS-ELM はその必要がない．また，新しく与えられた訓練データに対してのみ学習を行えばよい．BP-NN と比較して小さな計算量で高速に学習できることが知られている [18]．

2.3　オートエンコーダを用いた教師なし異常検知

オートエンコーダ [11] とは，ニューラルネットワークを用いた次元圧縮アルゴリズムの 1 つである．図 **3** にその学習手法を示す．オートエンコーダは入力データをそのまま教師データとして流用し，入力データを推論結果として再構成できるように学習する．このとき，隠れ層のノード数を入力層と出力層のノード数よりも小さく設定することで，入力データと推論結果の誤差が収束した場合に隠れ層行列を入力データの次元圧縮形式として見なせる．つまり，入力データ $\boldsymbol{x}$ のエンコード結果は $\boldsymbol{H} = G(\boldsymbol{x} \cdot \boldsymbol{\alpha} + \boldsymbol{b})$ として得られ，$\boldsymbol{H}$ のデコード結果は $\boldsymbol{y} = \boldsymbol{H} \cdot \boldsymbol{\beta}$ として得られる．また，オートエンコーダは別個に教師データを作成する必要がなく，入力データのみで学習が完結するため，教師なし学習アルゴリズムに分類される．

オートエンコーダは学習したことがない特徴を持つ入力データに対しては誤差が増加するが，教師なし異常検知モデルとして用いる際にはこの点を利用する．すなわち，まずオートエンコーダを正常データのみで学習させる．す

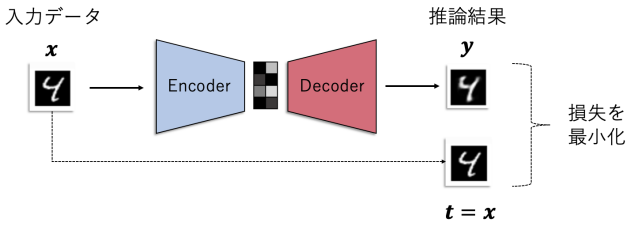


図 3　オートエンコーダ
Fig. 3　Autoencoder.

ると，正常データとは異なる特徴を持つデータ（異常データ）が与えられると相対的に誤差が増加するため，そこに閾値を設けることで異常データを検出できる．同様な手法として，オートエンコーダではなく PCA（Principal Component Analysis）を用いた手法があげられるが，オートエンコーダは PCA よりも異常検知の精度が高いことが知られている [24]．また，Kernal PCA [29] は非線形な入力データを扱うことができるが，PCA と比較して膨大な学習時間が消費される．一方でオートエンコーダは非線形な活性化関数を用いるだけで Kernel PCA より大幅に少ない計算時間で学習が完了する [24]．

2.4　教師なし異常検知における正常データの変動について

教師なし異常検知では正常データの分布を学習し，そこから離れたデータを異常データと見なす．この場合，異常検知の性能はいかに正確に正常データの分布を学習できるかが重要になるが，実世界のデータにおける Concept Drift と呼ばれる現象がそれを困難にする場合がある．Concept Drift とは，学習すべきデータの分布が時系列で変動する現象を指し [10]，教師なし異常検知の文脈では，それによって学習時に利用した正常データの分布と現在の正常データの分布に乖離が生じる．対象となる個体やデータの状態が時間要素によって動的に変化したり，それらの周囲の環境を十分に管理することが難しい場合に発生しやすく，たとえばユーザモデリング [30] やネットワークの監視，株価予測 [15] の際に発生することが知られている．また，それらのデータに対し教師なし異常検知を行う際には，正常データの分布に変動が生じることを考慮に入れる必要がある．

3.　関連研究

3.1　OS-ELM を用いた異常検知

実際の異常検知の現場では正常データの特徴は時間とともに変動する．それに対し，OS-ELM はそれらの変動に即時に追従できるため異常検知との親和性が高い．Singh らは OS-ELM ベースの IDS（Intrusion Detection System）を提案し，限定的なメモリ量で従来手法よりも高速かつ正確に大量のネットワークトラフィックから異常を検出できることを示した [26]．また，Bosman らは OS-ELM を用いたワイヤレスネットワークの教師なし異常検知手法を提案し，異常データを事前に用意する必要なく高い精度で異常が検出できることを示した [2]．本研究では上記先行研究と同様に OS-ELM を用い，さらにオートエンコーダを組み合わせることで，高速かつ高精度な教師なし異常検知手法を提案する．我々が把握する限り，OS-ELM とオートエンコーダを組み合わせた教師なし異常検知モデルは報告されていない．

3.2　OS-ELM のハードウェア実装

ELM のハードウェア実装に関する研究はすでに複数報告されている [7], [9], [32]．しかし，OS-ELM のハードウェア実装に関する研究はほとんど存在しない．Bosman らはハードウェア資源の限られた組み込み機器向けに OS-ELM を固定小数点で実装し，さらに計算精度の低下を防ぐための機構を提案した [3] が，実装はソフトウェアによって行われている．一方で，OS-ELM の FPGA 実装は報告されていない．

4.　OSUAD

本論文で提案する OSUAD は，OS-ELM とオートエンコーダを組み合わせた FPGA ベースの教師なし異常検知器である．4.1 節では OS-ELM の解析を基に OSUAD の学習処理の計算コストを削減する手法を提案し，4.2 節では学習処理と推論処理のアルゴリズムを示す．

4.1　OS-ELM の解析

OS-ELM の逐次学習式（式 (4)）は主に，行列積と逆行列演算の 2 種類で構成される．ここで，行列積 $\boldsymbol{A} \in \boldsymbol{R}^{p\times q} \cdot \boldsymbol{B} \in \boldsymbol{R}^{q\times r}$ にかかる計算イテレーション数を pqr，逆行列演算 $\boldsymbol{C}^{-1} \in \boldsymbol{R}^{r\times r}$ にかかる計算イテレーション数を r^3 とすると，式 (4) に存在する行列積と逆行列演算の合計の計算イテレーション数はそれぞれ以下のように求まる．

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

I_{prod} は行列積の，I_{inv} は逆行列演算の合計の計算イテレーション数を示す．また k は訓練データのバッチサイズであり，n，$\tilde{N}$，m はそれぞれ入力層，隠れ層，出力層のノード数である．例として，$\boldsymbol{H}_i\boldsymbol{P}_{i-1}\boldsymbol{H}_i^T$ の場合，まず $\boldsymbol{H}_i \in \boldsymbol{R}^{k\times\tilde{N}} \cdot \boldsymbol{P}_{i-1} \in \boldsymbol{R}^{\tilde{N}\times\tilde{N}}$ を計算し，次に $\boldsymbol{H}_i\boldsymbol{P}_{i-1} \in \boldsymbol{R}^{k\times\tilde{N}} \cdot \boldsymbol{H}_i^T \in \boldsymbol{R}^{\tilde{N}\times k}$ を計算する．この場合，最終的な計算イテレーション数は $I = k\tilde{N}^2 + k^2\tilde{N}$ となる．

ここで，バッチサイズが k のときの式 (4) 中の行列積と逆行列演算の計算イテレーションの合計を I_k とおくと，以下が成立する．

$$
\begin{aligned}
I_k &= I_{prod} + I_{inv} \\
&= 4k\tilde{N}^2 + k(2k+2m+n)\tilde{N} + k^3 \\
&= k(4\tilde{N}^2 + (2k+2m+n)\tilde{N} + k^2) \\
&\geq k(4\tilde{N}^2 + (2+2m+n)\tilde{N} + 1) = kI_1 \qquad (7)
\end{aligned}
$$

上式により $I_k \geq kI_1$ が成立する．つまり，式 (4) の行列積と逆行列演算はバッチサイズ k の訓練データを 1 度学習するよりも，バッチサイズを 1 として k 回学習する方が全体の計算イテレーション数が小さくなる（または，等しくなる）．ソフトウェアで式 (4) を実装した場合，1 度の学習ごとに関数呼び出しやメモリ確保などのオーバヘッドが生じる可能性があるため，上記は必ずしも成立しない．しかし，本研究のように専用回路として実現する場合，少なくとも関数呼び出しのオーバヘッドは存在しないため，バッチサイズを 1 に固定することで計算量を削減できる．

さらに，式 (4) の計算量的ボトルネックは逆行列演算 $(\boldsymbol{I} + \boldsymbol{H}_i\boldsymbol{P}_{i-1}\boldsymbol{H}_i^T)^{-1}$ であるが，$(\boldsymbol{I} + \boldsymbol{H}_i\boldsymbol{P}_{i-1}\boldsymbol{H}_i^T)$ の行列サイズは $k \times k$ であるため，$k = 1$ のとき，逆行列演算を逆数演算に置き換えられる．よって，式 (4) は次のように変形できる．

$$
\begin{aligned}
\boldsymbol{P}_i &= \boldsymbol{P}_{i-1} - \frac{\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T\boldsymbol{h}_i\boldsymbol{P}_{i-1}}{1 + \boldsymbol{h}_i\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T} \\
\boldsymbol{\beta}_i &= \boldsymbol{\beta}_{i-1} + \boldsymbol{P}_i\boldsymbol{h}_i^T(\boldsymbol{t}_i - \boldsymbol{h}_i\boldsymbol{\beta}_{i-1})
\end{aligned} \qquad (8)
$$

$\boldsymbol{h} \in \boldsymbol{R}^{1\times\tilde{N}}$ は $k = 1$ の時の隠れ層行列 $\boldsymbol{H}$ である．よって，バッチサイズを 1 に固定することで計算量を小さくできるうえに，逆行列演算器に必要な実装コストやハードウェアの資源を削減できる．上記の考察をふまえ，OSUAD 内で使用する OS-ELM のバッチサイズは 1 とする．

4.2　アルゴリズム

OSUAD は OS-ELM とオートエンコーダを組み合わせることで教師なし異常検知を実現する．2.3 節で述べたように，オートエンコーダは入力データを教師データとしてそのまま利用するため，式 (8) において $\boldsymbol{t}_i = \boldsymbol{x}_i$ が成立する．よって，OSUAD の逐次学習式は最終的に以下の数式で表現される．

$$
\begin{aligned}
\boldsymbol{P}_i &= \boldsymbol{P}_{i-1} - \frac{\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T\boldsymbol{h}_i\boldsymbol{P}_{i-1}}{1 + \boldsymbol{h}_i\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T} \\
\boldsymbol{\beta}_i &= \boldsymbol{\beta}_{i-1} + \boldsymbol{P}_i\boldsymbol{h}_i^T(\boldsymbol{x}_i - \boldsymbol{h}_i\boldsymbol{\beta}_{i-1})
\end{aligned} \qquad (9)
$$

また，ある入力 $\boldsymbol{x}$ に対する損失値は，損失関数 L を用いて以下のように計算される．

$$
loss = L(\boldsymbol{x}, G(\boldsymbol{x}\cdot\boldsymbol{\alpha} + \boldsymbol{b})\boldsymbol{\beta}) \qquad (10)
$$

上式で計算される損失値が閾値を上回れば，$\boldsymbol{x}$ を異常データと見なせる．

たとえば，モータの振動データを用いて異常を検出することを想定した場合，モータ 1 つ 1 つに OSUAD をあてがう．そして，あらかじめ集めた正常データか，最初の一定の時間内に流れてくる入力データを正常データと見なし，それを用いて初期学習を行う．初期学習が完了したら，入力データに対して推論を行い，たとえば得られた損失値が閾値より高ければ学習せずログとしてその旨を報告し，そうでなければ正常データと見なして学習を行うといった使い方が考えられる．

4.3　バッチサイズと学習の安定性に関する議論

BP-NN では通常，16 以上などある程度大きなバッチサイズに設定して学習を行う [14], [24], [25]．これは，ある訓練データのバッチサイズを k とすると，BP-NN では k 個分の誤差に対する重みの勾配を計算し，それを集約（多くの場合平均をとる）したものを最適化アルゴリズムに適用するため，k の数が小さいと学習の収束が不安定になるためである．

一方，OS-ELM の場合はバッチサイズを小さくしたからといって，上記のような問題は起きない．なぜなら，2.2 節で述べたように，OS-ELM は ELM の学習式（式 (2)）を逐次的に計算できるように拡張したものであり，たとえば全 100 サンプルの訓練データが与えられたとすると，バッチサイズ 10 で 10 回学習したときと，バッチサイズ 1 で 100 回学習したときでは，同じ重み $\boldsymbol{\beta}$ が得られるためである．よって，OSUAD において OS-ELM のバッチサイズは 1 とするが，それによって特別学習が不安定になることはない．

しかし，OS-ELM においては，バッチサイズにかかわらず，式 (4) 中の $(\boldsymbol{I} + \boldsymbol{H}_i\boldsymbol{P}_{i-1}\boldsymbol{H}_i^T)$ が非正則，あるいは非正則に近い場合に，学習が不安定になるという問題がある．OSUAD においては式 (9) 中の $1 + \boldsymbol{h}_i\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T$ が 0 である場合に非正則となり，学習が実行できなくなる．そこで，OSUAD では $1 + \boldsymbol{h}_i\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T$ が非正則に近い場合，つまり，ある小さな正の定数 EPSILON に対し，EPSILON $> 1 + \boldsymbol{h}_i\boldsymbol{P}_{i-1}\boldsymbol{h}_i^T$ が成立した場合は学習を行わずに入力データを捨てることで，上記の問題に対処する．

5.　実装

OSUAD は高位合成を用いて実装を行った．ツールは Xilinx 社の Vivado HLS 2016.4 であり，対象の FPGA は Xilinx 社のローエンドモデルである XC7Z010-1CLG400C とする．また，FPGA の動作周波数は 100 MHz である．

5.1　全体図

図 4 に OSUAD のブロック図を示す．OSUAD は *seq_train* モジュールと *predict* モジュールの 2 つのサブモジュールで構成される．*seq_train* モジュールは与えられた入力データ $\boldsymbol{x}$ に対して式 (9) を計算し，重み $\boldsymbol{\beta}$ と $\boldsymbol{P}$ を更新する．一方で *predict* モジュールは式 (10) を計算し，入

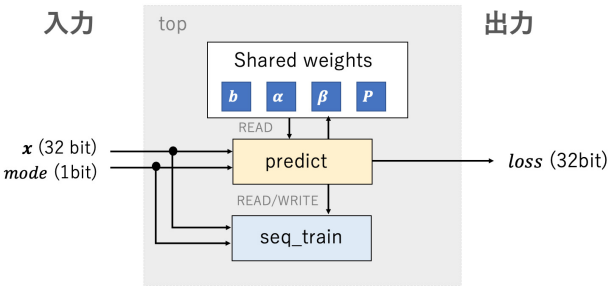


図 4　OSUAD のブロック図
Fig. 4　Block diagram of OSUAD.

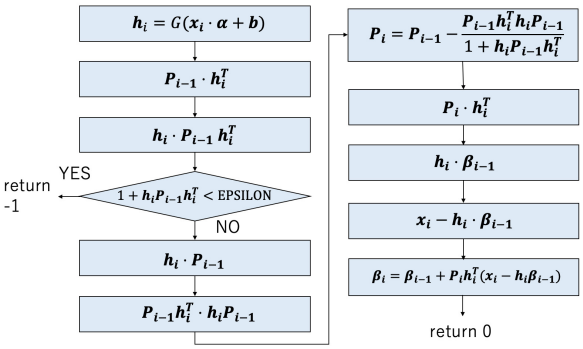


図 5　*seq_train* モジュールのフローチャート
Fig. 5　Flowchart of *seq_train* module.

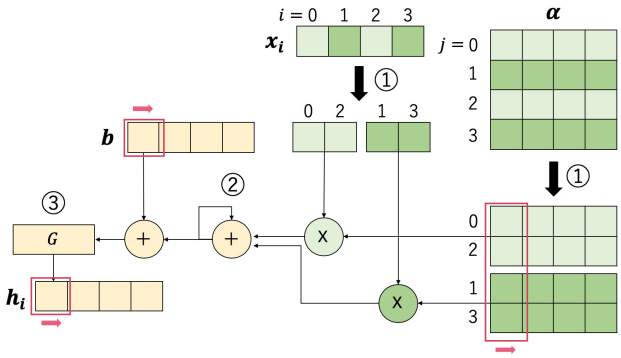


図 6　$h_i = G(x_i \cdot \alpha + b)$ の積和演算の並列実行
Fig. 6　Parallel execution of $h_i = G(x_i \cdot \alpha + b)$.

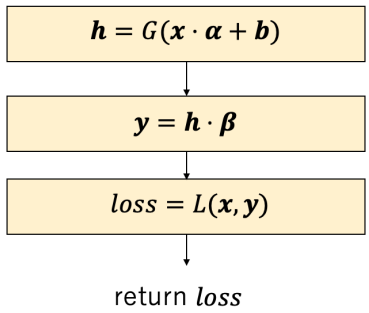


図 7　*predict* モジュールのフローチャート
Fig. 7　Flowchart of *predict* module.

力データの損失値を計算する．1 ビットの入力信号 *signal* は入力データに対して逐次学習を行うか，または推論処理を行うかを指定する．信号の値が 0 であれば *predict* モジュールに，1 であれば *seq_train* モジュールに入力データを渡す．また，両モジュール内のすべての実数は 32 ビットの固定小数点（整数部 10 ビット，小数部 22 ビット）を用いる．

5.2　*seq_train* モジュール

seq_train モジュールは式 (9) を計算する．**図 5** は同モジュールの処理フローを示す．図中の各処理ブロックは逐次的に実行される．4.3 節で述べたように，EPSILON $> 1 + \boldsymbol{h}_i \boldsymbol{P}_{i-1} \boldsymbol{h}_i^T$ が成立した場合は $\boldsymbol{\beta}$ と $\boldsymbol{P}$ の更新を行わずに入力データを捨てる．

さて，OSUAD の逐次学習処理の大部分は行列積，あるいはベクトル積によって占められる．これらの演算は積和演算部分を N 並列で実行することにより容易に高速化できる．ここでは，例として $\boldsymbol{h}_i = G(\boldsymbol{x}_i \cdot \boldsymbol{\alpha} + \boldsymbol{b})$ 中の積和演算を $N = 2$ 並列で実行する手法を示す．簡単のため，入力層のノード数と隠れ層のノード数を 4 とする．このとき，$\boldsymbol{x}_i$，$\boldsymbol{\alpha}$，$\boldsymbol{h}_i$ のそれぞれのサイズは 1×4，4×4，1×4 になる．

図 6 に並列実行の手順を示す．① $N = 2$ 並列で積和演算を実行するには $\boldsymbol{x}_i$ と $\boldsymbol{\alpha}$ のそれぞれを $N = 2$ つの部分行列に分割し，$N = 2$ つの値を同時に読み出す必要がある．そのために，まず $\boldsymbol{x}_i/\boldsymbol{\alpha}$ の列/行にインデックス i/j を割り当てる．そして $i/j \mod N = 2$ の計算結果が同じ列/行どうしを 1 つの部分行列として統合する．②上記で分割したそれぞれの部分行列から同時に値を読み出し，積和演算を行う．また，ここで得られた演算結果は総和として蓄積される．この処理は図中 $\boldsymbol{\alpha}$ の赤枠内のすべての値が読み出されるまで繰り返される．③ $\boldsymbol{b}$ の赤枠の値を読み出し，上記で得られた総和に足しこむ．そして，総和を 0 に戻し，得られた計算結果に対して活性化関数 G を適用する．最後に，$\boldsymbol{h}_i$ の赤枠の要素に出力値を書き込み，すべての赤枠を右に 1 列分スライドさせる．④すべての赤枠内にベクトルの要素値，あるいは行列の列が存在する場合，②に戻る．上記並列実行手法は消費されるハードウェア量が N に比例するため，高速化の倍率と増加するハードウェア量のトレードオフを考慮して調節する必要がある．

5.3　*predict* モジュール

predict モジュールは式 (10) を計算し，入力データの損失値を出力する．**図 7** は同モジュールの処理フローを示す．図中の各処理ブロックは逐次的に実行される．式 (10) 中の損失関数 L は二乗平均誤差など様々な関数が適用可能であるが，OS-ELM は学習結果が損失関数の種類に非依存であるため，できるだけ少ないハードウェア資源量で実装可能なもの（たとえば絶対平均誤差）を選ぶのが良い．また *seq_train* モジュールと同様，*predict* モジュールも演算

表 1　評価マシン
Table 1　Evaluation machine.

OS	Ubuntu 16.04 LTS
CPU	Intel Core i7-6700 3.5 GHz
GPU	Nvidia GTX 1070 VRAM 8 GB
DRAM	DDR4 RAM 32 GB
Storage	SSD 1 TB

の大部分が行列積で占められるため，同様な手法で高速化を実現できる．

6. 評価

ここまで，OS-ELM とオートエンコーダを組み合わせた FPGA ベースの教師なし異常検知器（OSUAD）とその実装について述べた．本章では，レイテンシ，FPGA リソースの使用率，異常検知能力について OSUAD の評価を行う．全評価を通じて**表 1** のマシンを用いる．

6.1 比較対象

OSUAD を以下の 4 つの CPU および GPU 実装と比較する．

(1) **OS-ELM-CPU**：OS-ELM を用いたオートエンコーダの CPU 実装
(2) **OS-ELM-GPU**：上記の GPU 実装
(3) **BP-NN-CPU**：BP-NN を用いたオートエンコーダの CPU 実装
(4) **BP-NN-GPU**：上記の GPU 実装

CPU 実装の比較対象には TensorFlow（ver 1.6.0）[1] を，GPU 実装の比較対象には TensorFlow の GPU 実行機能を有効にしたものを用い，計算精度は 32 bit の浮動小数点を用いている．BP-NN-CPU と BP-NN-GPU は，OS-ELM ではなく BP-NN でオートエンコーダを構築したものを指す．

6.2 各実装の設定

まず OSUAD 含む全実装共通の設定として，オートエンコーダの各層のノード数は共通とし，バッチサイズは 1 とする．誤差関数は絶対平均誤差 $L(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n}|x_i - y_i|$ を用い，4.3 節の定数 EPSILON は 1e-4 とする．また，OSUAD，OS-ELM-CPU，OS-ELM-GPU の共通の設定として，活性化関数を恒等関数 $G(\boldsymbol{x}) = \boldsymbol{x}$ とする[*2]．OSUAD 内のすべての行列積，ベクトル積の並列数は $N = 2$ としている．BP-NN-CPU と BP-NN-GPU の共通の設定として，隠れ層の活性化関数として Relu 関数を [22]，出力層の活性化関数としてシグモイド関数を用いる．最適化アルゴリズムとしては Adam [16] を用い，ハイパーパラメータは文献 [16] の推奨値（学習係数 $= 0.001$，$\beta_1 = 0.9$，$\beta_2 = 0.999$）を採用している．

[*2] 実際には恒等関数に加え，シグモイド関数 [6] などの非線形関数も実装しているが，後述する異常検知能力の評価においてその差異が小さかったため実装容易性を考慮し，今回は恒等関数を選択している．

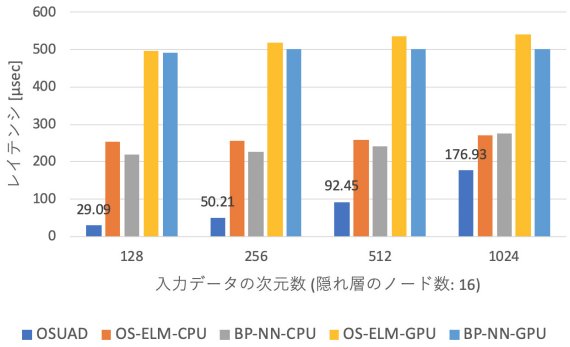


図 8　隠れ層のノード数 = 16 のときの学習レイテンシ
Fig. 8　Training latency ($\tilde{N} = 16$).

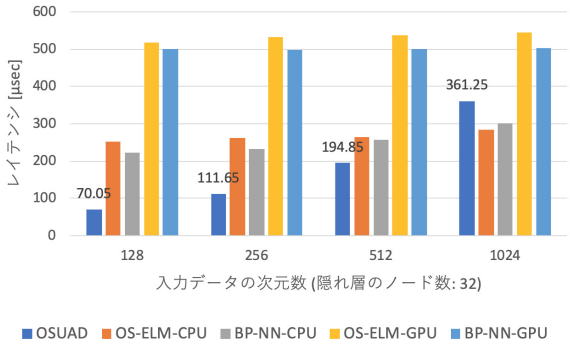


図 9　隠れ層のノード数 = 32 のときの学習レイテンシ
Fig. 9　Training latency ($\tilde{N} = 32$).

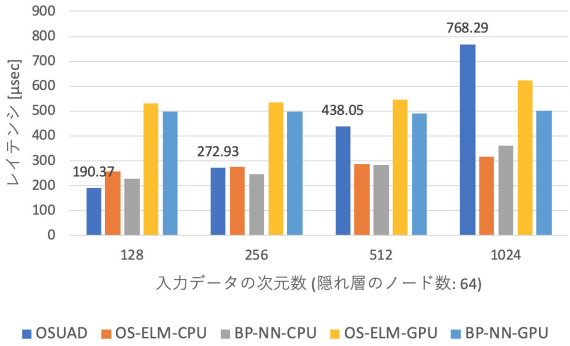


図 10　隠れ層のノード数 = 64 のときの学習レイテンシ
Fig. 10　Training latency ($\tilde{N} = 64$).

6.3 レイテンシ

6.3.1 学習レイテンシ

図 8，**図 9**，**図 10** はそれぞれ隠れ層のノード数が 16，32，64 の時の各実装の学習処理のレイテンシを示す．横軸は入力データの次元数であり，縦軸は入力データを受け取ってから学習が完了するまでのレイテンシである．

式 (7) より，学習レイテンシは入力データの次元数に

比例する．この時，CPU で実装された OS-ELM-CPU と BP-NN-CPU は入力データの次元数を増加させてもレイテンシがほとんど変化しないことから，学習の計算時間よりも関数呼び出しなどのオーバヘッドにかかる時間の方が支配的である．OSUAD はそれらオーバヘッドが存在しないため，特に入力データの次元数や隠れ層のノード数が小さい場合に CPU 実装よりも高速に実行できる．一方で，そうでない場合には CPU の方が圧倒的に動作周波数が高いため OSUAD の方が低速になる．また，GPU で実装された OS-ELM-GPU と BP-NN-GPU は，上記 2 つの CPU 実装よりもすべての場合において約 2 倍の実行時間がかかっているが，これはホストと GPU の通信時間によるものである．GPU は大きなバッチサイズのデータに対して並列計算する場合に効果的であるため，バッチサイズが 1 である OSUAD の実装プラットフォームとしては不適切である．結果として，OSUAD は OS-ELM-CPU，BP-NN-CPU，OS-ELM-GPU，BP-NN-GPU のそれぞれに対し平均 2.47 倍，2.24 倍，4.95 倍，4.74 倍の学習処理の高速化を実現した．

6.3.2　推論レイテンシ

図 11，**図 12**，**図 13** はそれぞれ隠れ層のノード数が 16，32，64 の時の各実装の推論処理のレイテンシを示す．横軸は入力データの次元数であり，縦軸は入力データを受け取ってから推論が完了するまでのレイテンシである．

OS-ELM-CPU と BP-NN-CPU の推論時の計算は基本的に同じである．しかし，前者は活性化関数が 1 つだけであるが，後者は隠れ層と出力層のそれぞれ 2 カ所に非線形関数を使用しているため若干計算量が増加する．また，推論時の計算量は学習時と同様に入力データの次元数に比例するため，同様な傾向が見られる．結果として，OSUAD は OS-ELM-CPU，BP-NN-CPU，OS-ELM-GPU，BP-NN-GPU のそれぞれに対し平均 3.77 倍，4.00 倍，6.60 倍，6.99 倍の推論処理の高速化を実現した．

本研究では OSUAD の行列積，ベクトル積の並列数は $N = 2$ としているが，6.4 節の FPGA リソースの使用率を考慮しつつその値を調節することで，推論・学習共にさらなる高速化を実現できる．

6.4　FPGA リソースの使用率

表 2 に，OSUAD の隠れ層のノード数をそれぞれ 16，32，64 とした時の各 FPGA リソースの使用率を示す．これらの値はすべて Vivado HLS 2016.4 の合成結果として得られたものである．実装対象の XC7Z010-1CLG400C は Xilinx 社のローエンドモデル FPGA であるが，最もモデルのサイズが大きい場合（隠れ層のノード数 = 64，入力データの次元数 = 1,024）を除くすべての条件においてリソース不足を起こすことなく実装できている．カラー画像など次元数が 10,000 を超える超高次元データに対応させるにはこの規模の FPGA では不十分であるが，100 から 1,000 次元程度であれば OSUAD は実用的なハードウェア量で実装できる．また，レイテンシをそれほど求めない場合，OSUAD の行列積，ベクトル積の並列数 N を調節することでさらに小さく実装可能である．

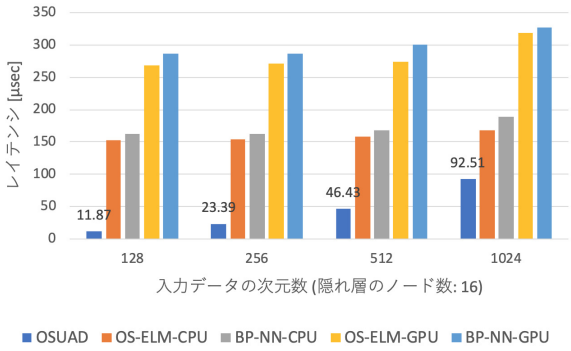


図 11　隠れ層のノード数 = 16 のときの推論レイテンシ
Fig. 11　Prediction latency ($\tilde{N} = 16$).

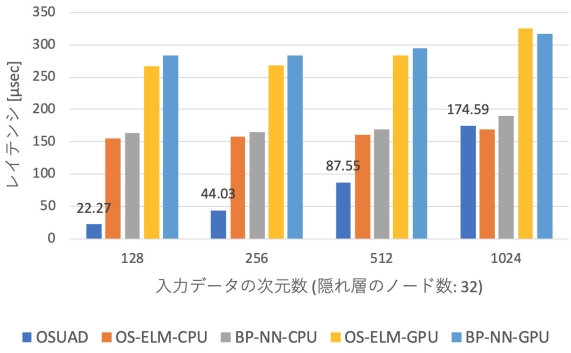


図 12　隠れ層のノード数 = 32 のときの推論レイテンシ
Fig. 12　Prediction latency ($\tilde{N} = 32$).

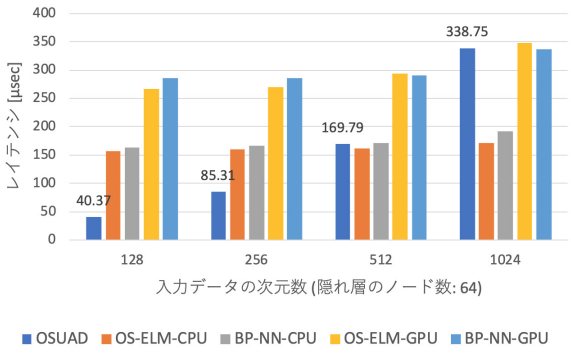


図 13　隠れ層のノード数 = 64 のときの推論レイテンシ
Fig. 13　Prediction latency ($\tilde{N} = 64$).

6.5　異常検知能力

6.5.1　評価手順

本章では次の手順で各実装の異常検知能力の評価を行う．①各実装を正常な訓練データを用いて学習させ，正常なテストデータに対する損失値を計算する．② ①で計算

表 2　OSUAD の各 FPGA リソースの使用率
Table 2　FPGA resource utilization of OSUAD.

隠れ層のノード数 = 16				
入力次元数	BRAM [%]	DSP [%]	FF [%]	LUT [%]
128	10	43	3	9
256	13	43	3	10
512	20	43	3	10
1,024	33	43	4	10
隠れ層のノード数 = 32				
入力次元数	BRAM [%]	DSP [%]	FF [%]	LUT [%]
128	14	43	3	10
256	20	43	3	10
512	34	43	3	10
1,024	60	43	3	10
隠れ層のノード数 = 64				
入力次元数	BRAM [%]	DSP [%]	FF [%]	LUT [%]
128	32	43	3	10
256	45	43	3	10
512	72	43	3	10
1,024	125	43	3	11

した損失値の平均値 μ と標準偏差 σ を計算する．③正常なテストデータと異常データを混合させたデータセットに対して損失値を計算する．あるデータの損失値を $loss$ とすると $\frac{loss-\mu}{\sigma} > \theta$ が成立したとき，該当するデータを異常データと見なす．式中の θ は異常度の閾値である．本評価では正常データの損失値が正規分布に従うと仮定し，損失値を平均と標準偏差で正規化した値を異常度とする．

評価指標としては**精度**，**再現率**，**F 値**を用いる．精度は異常と判定されたデータのうち実際に異常であるものの割合を示し，再現率は全異常データのうち異常であると判定されたものの割合を示す．F 値は上記 2 つの指標の調和平均であり，総合的な異常検知の性能を示す．これら 3 つの指標をそれぞれ p，r，f とおくと，以下のように計算される．

$$p = \frac{TP}{TP+FP}, r = \frac{TP}{TP+FN}, f = \frac{2pr}{p+r} \tag{11}$$

式 (11) 中の TP，FP，FN はそれぞれ True Positive（異常データに対し異常と予測した数），False Positive（正常データに対し異常と予測した数），False Negative（異常データに対し正常と予測した数）を意味する．また，上式には登場しないが True Negative は正常データに対し正常と予測した数である．精度あるいは再現率だけでは，FP と，FN を同時に評価できないが，F 値は精度と再現率の調和平均であるため，それらを同時に評価する総合的な指標として利用される [13]．

比較対象について，CPU 実装と GPU 実装は同じ単精度浮動小数点を用いており，異常検知精度に差異が生じないため，OS-ELM-GPU と BP-NN-GPU を除外し，OS-ELM-CPU と BP-NN-CPU を代表として用いる．また，本評価

図 14　MNIST のサンプル画像
Fig. 14　Sample images of MNIST.

図 15　Fashion-MNIST のサンプル画像
Fig. 15　Sample images of Fashion-MNIST.

においては BP-NN-CPU を深層化した場合の精度を評価するために，新たに Deep-BP-NN を導入する．Deep-BP-NN は BP-NN-CPU を 3 層から 7 層に深層化させたものであり，各層のノード数はそれぞれ入力層から 784，128，64，32，64，128，784 とし，活性化関数は出力層がシグモイド関数，それ以外を Relu 関数とする．その他の設定（損失関数，最適化アルゴリズム）は BP-NN-CPU と同じとする．近年の BP-NN を用いた教師なし異常検知手法では多くの場合深層化されたモデルを用いているため [14], [25], [33]，Deep-BP-NN と OSUAD を比較することで，OSUAD が近年の設定に対しどの程度の精度を有するかを評価する．

6.5.2　データセット

本評価では正常データとして MNIST [17] を用いる．**図 14** はそのサンプル画像である．MNIST は 28×28 の 10 クラスの手書き数字画像データセットであり，60,000 枚の訓練データと 10,000 枚のテストデータで構成される．本評価では前者を正常な訓練データとして，後者を正常なテストデータとして用いる．

異常データとしては Fashion-MNIST [31] を用いる．**図 15** はそのサンプル画像である．Fashion-MNIST は MNIST と同様に 28×28 の 10 クラスの衣類画像のデータセットであり，同じく 60,000 枚の訓練データと 10,000 枚のテストデータで構成される．本評価では後者のみを異常データとして用いる．

上記すべての画像データは画素値を 255 で割ることで [0,1] に正規化し，$28 \times 28 = 784$ 次元ベクトルに変換したものを入力データとして扱う．

6.5.3　設定

各実装のモデルの入力層と出力層のノード数は 784 次元とし，隠れ層のノード数は 32 とする．表 2 より，上記サイズの OSUAD はリソース不足なしで実装可能である．初期化処理としては，OS-ELM-CPU の重み $\boldsymbol{\alpha}$ を [0,1] の一様分布で生成する．また，式 (5) を用いて $\boldsymbol{P}_0$ と $\boldsymbol{\beta}_0$ を計算する．これらを計算する際には，隠れ層のノード数以上の訓練データが必要であるため [18]，隠れ層のノード数 ×1.2 枚の画像を正常な訓練データからサンプリングする．OSUAD は上記によって得られた $\boldsymbol{\alpha}$，$\boldsymbol{P}_0$，$\boldsymbol{\beta}_0$ を利用する．

学習手順として，OSUAD，OS-ELM-CPU は正常な訓

表 3　各実装の異常検知能力の比較
Table 3　Comparison of anomaly detection accuracy.

実装	θ	精度	再現率	F 値
OSUAD	1.0	0.853	0.922	0.886
OS-ELM-CPU	1.0	0.856	0.926	0.890
BP-NN-CPU	1.0	0.855	0.913	0.883
Deep-BP-NN	1.0	0.859	0.961	0.907
OSUAD	2.0	0.966	0.855	0.907
OS-ELM-CPU	2.0	0.971	0.850	0.911
BP-NN-CPU	2.0	0.958	0.812	0.879
Deep-BP-NN	2.0	0.956	0.900	0.927
OSUAD	3.0	0.996	0.770	0.869
OS-ELM-CPU	3.0	0.996	0.766	0.866
BP-NN-CPU	3.0	0.992	0.678	0.805
Deep-BP-NN	3.0	0.992	0.811	0.892

練データを 1 エポック学習する．これは 2.2 節で述べたように，OS-ELM はあるデータセットに対して 1 度学習するだけで最適解が得られるためである．一方で BP-NN-CPU は繰り返し型の最適化手法である Backpropagation 法を採用するため，同じデータセットに対し 1 エポック学習するだけでは十分な精度が得られなかった．そのため，これら 2 つの実装については 10 エポック学習を行う．また，Backpropagation 法による最適化はバッチサイズが小さいと学習の収束が不安定になり，精度が悪化するため特別にバッチサイズを 64 とした．

6.5.4　評価結果

表 3 に評価結果を示す．OSUAD の最大の F 値は，BP-NN-CPU と比較して 2.4%高い．また OSUAD は 1 エポックしか学習していないのに対し，BP-NN-CPU は 10 エポック学習しているため，提案実装は 10 分の 1 のエポック数でより高い性能を実現している．さらに BP-NN-CPU は，性能を向上させるためにバッチサイズを 1 から 64 に拡張しているため，OSUAD はより小さいバッチサイズで高い性能が得られている．4.3 節で述べたように，OS-ELM はバッチサイズを変化させても最終的に得られる $\boldsymbol{\beta}$ は同一であるため，OSUAD はバッチサイズが 1 でも学習の収束が不安定にならない．しかし，BP-NN-CPU の深層版である Deep-BP-NN については，OSUAD における最大の F 値よりも 2.0%高い値が得られた．一般的に層の深いモデルは浅いモデルに比べて表現能力や精度が向上することが知られているが，本評価においても同様な傾向が確認された．このように BP-NN は異常検知においても深層化することでより高い性能が得られるが，OS-ELM は 3 層に限定されたモデルであるため同様な手法は使えない．よってこの点は BP-NN に対する弱みである．一方，OSUAD は小規模な FPGA 上にオンラインに逐字的に学習ができるという利点がある．Deep-BP-NN との最大の F 値の差は 2.0%となったが，上記の利点が優先される場合は OSUAD は有力な選択肢の 1 つとなる．

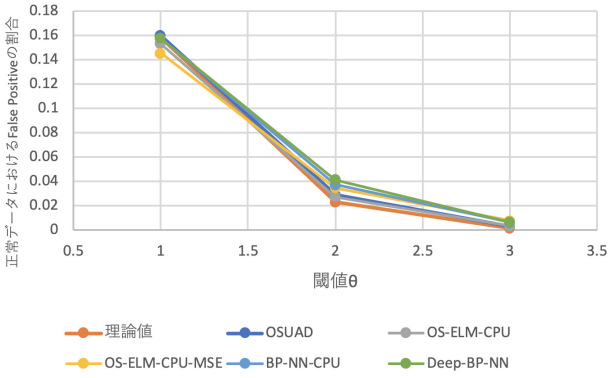


図 16　閾値 θ と正常データの False Positive の割合
Fig. 16　Relationship between θ and False Positive rate.

また，**図 16** に表 3 における各実装の閾値 θ と正常データに対する False Positive の割合を示す．凡例の「理論値」は，正常データの損失値が正規分布であると仮定した場合の理論値である．6.5.1 項で述べたように，本評価では正常データの損失値が正規分布に従うと仮定しているため，その仮定が正しければ，各実装の False Positive の割合は理論値に近づくはずである．結果として，OSUAD の False Positive の割合は，それぞれ $\theta = 1$，$\theta = 2$，$\theta = 3$ の場合，理論的には 15.8%，2.28%，0.135%が期待されるが，実際にはそれぞれ 15.9%，3.01%，0.309%が得られ，大きな分布の乖離は見られなかった．また，「OS-ELM-CPU-MSE」は OS-ELM-CPU の損失関数（絶対平均誤差）を二乗平均誤差に変更し，評価を取ったものであるが，この実装においても同様な結果が得られたため，少なくとも本評価で用いたデータセットにおいては損失関数の種類によらず，仮定した分布に近い分布が得られている．また，BP-NN-CPU や Deep-BP-NN においても同様な結果が得られている．

なお，OSUAD，OS-ELM-CPU は同じ重み $\boldsymbol{\alpha}$ と $\boldsymbol{P}_0$，$\boldsymbol{\beta}_0$ を共有するため理想的にはまったく同じ結果になるはずであるが，OSUAD は 32 bit の固定小数点を，OS-ELM-CPU は 32 bit の浮動小数点を用いているため僅かに計算結果が異なる．また OSUAD の学習過程において，4.3 節のデータを捨てる処理が発生することはなかった．

7.　まとめ

本研究では，FPGA を用いたオンライン逐次学習型教師なし異常検知器（OSUAD）を提案した．OSUAD はニューラルネットワークの応用技術の 1 つであるオートエンコーダと，高速なオンライン逐次学習アルゴリズムの OS-ELM を組み合わせることで，事前に異常データを集める必要なしで高速に異常データを検知できる．また，OS-ELM はバッチサイズを 1 に固定することで，計算量の点でボトルネックである逆行列計算を完全に消去できる点に着目し，学習処理の高速化と省面積化を実現した．OSUAD は正常

データの特徴の変動に即時的に追従することができ，かつ置かれた環境に特化したモデルを学習できる．本研究は，オートエンコーダと OS-ELM の組合せを専用ハードウェアとして実現した最初の研究である．

評価結果として，OSUAD は CPU と GPU で実装したものと比較してそれぞれ平均 2.47 倍，4.95 倍高速な学習処理を実現し，平均 3.77 倍，6.60 倍高速な推論処理を実現した．またオープンソースのデータセットを用いた評価を行った結果，OSUAD は現在主流の Backprogation 法を用いたニューラルネットワーク（BP-NN）を用いた実装と比較して，10 分の 1 の学習エポック数で 2.4%高い F 値を実現した．また，深層化した BP-NN に対しても 2.0%の F 値の差に抑えられた．

今後の展望としては，忘却機能の付与が考えられる．現状の提案手法では忘却手法がなく，時系列で正常データの特徴が変化した際に，過去に正常データとして学習したデータが異常検知の精度に悪影響を与える可能性があるため，直近に学習したデータに対して大きな重み付けするような何らかの手法が必要である．また，OS-ELM は全結合層で構成されたニューラルネットワークであり，そのままでは Backpropagation 法を用いたモデルのように畳み込み層を積層することはできない．そのため，現状の提案手法は 6.5 節で用いたような 1,000 次元未満程度のデータに対しては異常検知を行うことができるが，数万～数百万次元で表現されるカラー画像や動画データに対して適用するのは困難である．上記の問題に対処するため，複数の OSUAD インスタンスを用いることが考えられる．その他に，6.5.4 項において正常データの損失値の分布に関する議論を行ったが，今回は単一のデータセットを用いており，他のデータセットに対しても真であるかは断定できないため，今後の課題とする．

参考文献

[1] Abadi, M. et al.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, *Proc. USENIX Conference on Operating Systems Design and Implementation*, pp.265–283 (2016).

[2] Bosman, H., Iacca, G., Tejada, A., Wörtche, H. and Liotta, A.: Spatial Anomaly Detection in Sensor Networks using Neighborhood Information, *Information Fusion*, Vol.33, pp.41–56 (2017).

[3] Bosman, H., Liotta, A., Iacca, G. and Wörtche, H.: Online Extreme Learning on Fixed-Point Sensor Networks, *Proc. IEEE International Conference on Data Mining Workshops*, pp.319–326 (2013).

[4] Breunig, M., Kriegel, H., Ng, R. and Sander, J.: LOF: Identifying density-based local outliers, *Proc. ACM Special Interest Group on Management of Data*, pp.93–104 (2000).

[5] Chen, Y., Zhou, X. and Huang, T.: One-class SVM for learning in image retrieval, *Proc. IEEE International Conference on Image Processing*, pp.34–37 (2001).

[6] Cybenko, G.: Approximation by Superpositions of a Sigmoidal Function, *Mathematics of Control, Signals and Systems*, Vol.2, No.4, pp.303–314 (1989).

[7] Decherchi, S., Gastaldo, P., Leoncini, A. and Zunino, R.: Efficient Digital Implementation of Extreme Learning Machines for Classification, *IEEE Trans. Circuits and Systems II: Express Briefs*, Vol.59, No.8, pp.496–500 (2012).

[8] Erfani, S., Rajasegarar, S., Karunasekera, S. and Leckie, C.: High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning, *Pattern Recognition*, Vol.58, pp.121–134 (2016).

[9] Frances, V. et al.: Hardware Implementation of Real-time Extreme Learning Machine in FPGA: Analysis of Precision, Resource Occupation and Performance, *Computers & Electrical Engineering*, Vol.51, pp.139–156 (2016).

[10] Hind, M., Mehta, S., Mojsilovic, A., Nair, R., Ramamurthy, K., Olteanu, A. and Varshney, K.: Increasing Trust in AI Services through Supplier's Declarations of Conformity, *CoRR*, Vol.abs/1808.07261 (2018).

[11] Hinton, G. and Salakhutdinov, R.: Reducing the Dimensionality of Data with Neural Networks, *Science*, Vol.313, No.5786, pp.504–507 (2006).

[12] Huang, G., Zhu, Q. and Siew, C.: Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks, *Proc. International Joint Conference on Neural Networks*, pp.985–990 (2004).

[13] 井手　剛，杉山　将：異常検知と変化検知，pp.8–11, 講談社 (2015).

[14] Jinwon, A. and Sungzoon, C.: Variational autoencoder based anomaly detection using reconstruction probability, *Special Lecture on IE*, Vol.2, pp.1–18 (2013).

[15] Kadwe, Y. and Suryawanshi, V.: A Review on Concept Drift, *IOSR Journal of Computer Engineering*, Vol.17, No.1, pp.20–26 (2015).

[16] Kingma, D. and Ba, J.: Adam: A Method for Stochastic Optimization, *CoRR*, Vol.abs/1412.6980 (2014).

[17] Lecun, Y. and Cortes, C.: MNIST handwritten digit database (2010), available from ⟨http://yann.lecun.com/exdb/mnist/⟩.

[18] Liang, N., Huang, G., Saratchandran, P. and Sundararajan, N.: A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks, *IEEE Trans. Neural Networks*, Vol.17, No.6, pp.1411–1423 (2006).

[19] Marco, G. and Alberto, T.: On the Problem of Local Minima in Backpropagation, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.14, No.1, pp.76–86 (1992).

[20] Montalvo, A., Gyurcsik, R. and Paulos, J.: Toward a general-purpose analog VLSI neural network with on-chip learning, *IEEE Trans. Neural Networks*, Vol.8, No.2, pp.413–423 (1997).

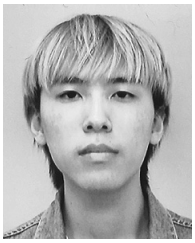
[21] Morie, T. and Amemiya, Y.: An all-analog expandable neural network LSI with on-chip backpropagation learning, *IEEE Journal of Solid-State Circuits*, Vol.29, No.9, pp.1086–1093 (1994).

[22] Nair, V. and Hinton, G.: Rectified Linear Units Improve Restricted Boltzmann Machines, *Proc. International Conference on Machine Learning*, pp.807–814 (2010).

[23] Reynolds, D.: *Encyclopedia of biometrics*, pp.659–663, Springer US (2009).

[24] Sakurada, M. and Yairi, T.: Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction,

Proc. Workshop on Machine Learning for Sensory Data Analysis, pp.4–11 (2014).
[25] Schlegl, T., Seeböck, P., Waldstein, S., Erfurth, U. and Langs, G.: Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery, *Proc. International Conference on Information Processing in Medical Imaging*, pp.146–157 (2017).
[26] Singh, R., Kumar, H. and Singla, R.: An Intrusion Detection System using Network Traffic Profiling and Online Sequential Extreme Learning Machine, *Expert Systems with Applications*, Vol.42, No.22, pp.8609–8624 (2015).
[27] Tsukada, M., Kondo, M. and Matsutani, H.: OS-ELM-FPGA: An FPGA-Based Online Sequential Unsupervised Anomaly Detector, *Proc. International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms* (2018). available from ⟨http://www.arc.ics.keio.ac.jp/~matutani/papers/tsukada_heteropar2018.pdf.⟩
[28] Verleysen, M. and François, D.: The Curse of Dimensionality in Data Mining and Time Series Prediction, *Proc. International Work-Conference on Artificial Neural Networks*, pp.758–770 (2005).
[29] Wang, Q. et al.: Kernel Principal Component Analysis, *Proc. International Conference on Artificial Neural Networks*, pp.583–588 (1997).
[30] Webb, G., Pazzani, M. and Billsus, D.: Machine Learning for User Modeling, *User Modeling and User-Adapted Interaction*, Vol.11, No.1-2, pp.19–29 (2001).
[31] Xiao, H., Rasul, K. and Vollgraf, R.: Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms (2017), available from ⟨https://github.com/zalandoresearch/fashion-mnist.⟩
[32] Yeam, T., Ismail, N., Mashiko, K. and Matsuzaki, T.: FPGA Implementation of Extreme Learning Machine System for Classification, *Proc. IEEE Region 10 Conference*, pp.1868–1873 (2017).
[33] Zhou, C. and Paffenroth, C.: Anomaly Detection with Robust Deep Autoencoders, *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.665–674 (2017).



塚田 峰登

2018 年慶應義塾大学理工学部情報工学科卒業．現在，同大学大学院理工学研究科開放環境科学専攻前期博士課程在籍．

近藤 正章　（正会員）

1998 年筑波大学第三学群情報学類卒業．2000 年同大学大学院工学研究科博士前期課程修了．2003 年東京大学大学院工学系研究科先端学際工学専攻修了．博士（工学）．独立行政法人科学技術振興機構戦略的創造研究推進事業 CREST 研究員，2004 年東京大学先端科学技術研究センター特任助手，2007 年同特任准教授，2008 年電気通信大学大学院情報システム学研究科准教授を経て，現在東京大学大学大学院情報理工学系研究科准教授．東京大学情報基盤センター，および理化学研究所計算科学研究センター兼務．計算機アーキテクチャ，ハイパフォーマンスコンピューティング，ディペンダブルコンピューティングの研究に従事．電子情報通信学会，IEEE，ACM 各会員．

松谷 宏紀　（正会員）

2004 年慶應義塾大学環境情報学部卒業．2008 年同大学大学院理工学研究科後期博士過程修了．博士（工学）．2009 年度～2010 年度まで，東京大学大学院情報理工学系研究科特別研究員，日本学術振興会特別研究員（SPD）．2011 年度慶應義塾大学理工学部情報工学科専任講師．2017 年度より同准教授．コンピュータアーキテクチャとインターコネクトに関する研究に従事．IEEE，電子情報通信学会各会員．