一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
VLD2018-86,CPSY2018-96,RECONF2018-60(2019-01)

オンライン逐次学習による教師なし異常検知コアの面積性能評価

井坪　知也†　塚田　峰登††　松谷　宏紀†

† 慶應義塾大学 理工学部　〒 223-8522 神奈川県横浜市港北区日吉 3-14-1
†† 慶應義塾大学大学院 理工学研究科　〒 223-8522 神奈川県横浜市港北区日吉 3-14-1
E-mail: †{itsubo,tsukada,matutani}@arc.ics.keio.ac.jp

あらまし　IoT 機器の普及によってエッジデバイスにおける異常検知の需要が高まっている。実世界では正常パターンが変動することがあり、そのような場合異常検知モデルの構築に使用した教師データと実際の正常パターンとの間の乖離が問題となる。このための解決策として、エッジデバイス上で異常検知モデルを学習するオンデバイス学習の利用が考えられる。しかし、デバイスで学習を行うためにはデバイスの面積等のコストの制約を満たさなければならない。本論文ではオンデバイス学習のために逐次学習アルゴリズム OS-ELM(Onlene Sequential Extreme Learning Machine) を基にした異常検知器を採用するが、IoT 機器を前提にオンデバイス学習を行うには面積等のコスト制約がシビアである。本論文では、オンデバイス学習器の設計空間探索を目的として、固定小数の bit 幅を変えた場合、除算器を bit シフトを用いて作成した場合と Newton 法を利用して作成した場合、演算器の使用回数を変化させた場合について Verilog HDL で実装し、評価を行うことで面積と実行時間、スループットのトレードオフを明らかにした。
キーワード　機械学習、AI チップ、オンデバイス学習

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

Tomoya ITSUBO†, Mineto TSUKADA††, and Hiroki MATSUTANI†

† Faculty of Science and Technology, Keio University 3-14-1, Hiyoshi,Yokohama, JAPAN 223-8522
†† Graduate School of Science and Technology, Keio University 3-14-1, Hiyoshi,Yokohama, JAPAN 223-8522
E-mail: †{itsubo,tsukada,matutani}@arc.ics.keio.ac.jp

1. はじめに

近年、工場やデータセンタなどの実世界での IoT 機器の利用が広まっている。IoT 機器を利用する重要なアプリケーションの一つとして異常検知が挙げられる。しかし、実環境では正常パターンが変動することがあり、そのような場合異常検知モデルの構築に使用した教師データと実際の正常パターンとの間の乖離 (コンセプトドリフト) が問題となる。環境ごとに正常パターンが異なる場合、環境ごとに教師データを事前に収集し、環境ごとに学習を行うことは煩雑であり、時として、正常データの収集が実世界における異常検知の足かせとなっている。このための解決策として、エッジデバイス上で異常検知モデルを学習するオンデバイス学習の利用が考えられる。

通常、学習を行うためには多くの計算リソースを必要とするが、エッジデバイスは面積や電力などの制約が大きい。塚田ら [1] は OS-ELM(Online Sequential Extreme Learning Machine) [2] ベースでオンデバイス学習を行うことのできる異常検知器を提案している。しかし、この異常検知器を用いてオンデバイス学習を行う際の面積や電力などのコストと性能のトレードオフに関して十分に検討されていない。本論文では、この異常検知器を固定小数の bit 幅を変えた場合、除算器を bit シフトを用いて作成した場合と Newton 法を利用して作成した場合、演算器の使用回数を変化させた場合、パイプラインを最適化させた場合について Verilog HDL で実装、評価を行うことにより面積と実行時間、スループットのトレードオフを明らかにした。

本論文の構成は、以下の通りである。2. 章で OS-ELM のアルゴリズムとその計算ボトルネックの解消法、オートエンコーダを利用した教師なし異常検知についての解説を行う。3. 章で今回扱った異常検知器の設計パラメータと設計オプションについての解説を行い、4. 章でそれぞれの設計を面積、実行周波数、電力に関して評価する。最後に、5. 章で本論文のまとめと今後の課題について述べる。

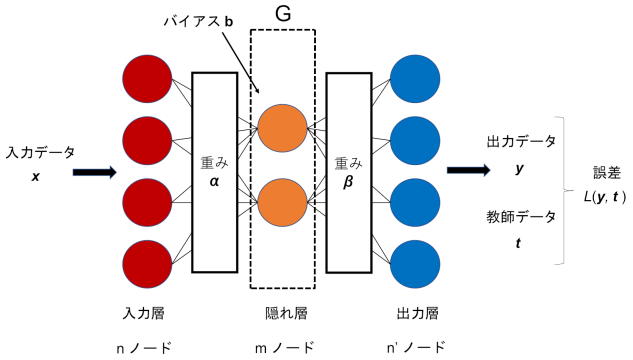

図 1　ELM

2. 関 連 研 究

2.1 オンデバイス学習

ProjectionNet [3] は Backpropagation 法を使用して LSTM 等の既存のアーキテクチャと計算コストの低い推論器を共同で学習することにより、デバイス上で低い計算コストの推論を行うことを提案している。Abdullah [4] らはメモリスタを使用することにより、Spiking Neural Network をオンデバイス学習向きに実装することを提案している。

塚田ら [1] は、高速な逐次学習アルゴリズムである OS-ELM とオートエンコーダを組み合わせたデバイス上で逐次学習できる FPGA ベースの教師なし異常検知器 OSUAD(Online Sequential Learning Unsupervised Anomaly Detector) を提案している。

本論文では、推論だけでなく学習もオンデバイスで行うことのできる OSUAD に注目し、OSUAD を様々な設計オプションで実装し、オンデバイス学習を行う際のコストと性能のトレードオフを明らかにする。

2.2 OSUAD

本節では、まず OS-ELM の前提知識として、まず ELM (Extreme Learning Machine) [5] について述べる。次に、OS-ELM のアルゴリズムについて述べる。2.2.3 節で塚田ら [1] による OS-ELM の計算量の削減法について述べ、2.2.4 節でオートエンコーダを利用した異常検知について述べる。

2.2.1 ELM

ELM はニューラルネットワークの一つであり、図 1 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 n' 次元の推論結果 $\boldsymbol{y} \in \boldsymbol{R}^{k\times n'}$ は、$\boldsymbol{y} = G(\boldsymbol{x}\cdot\boldsymbol{\alpha}+\boldsymbol{b})\boldsymbol{\beta}$ として得られる。この時、$\boldsymbol{\alpha} \in \boldsymbol{R}^{n\times m}$ は入力層と隠れ層を結合する重みであり、$\boldsymbol{\beta} \in \boldsymbol{R}^{m\times n'}$ は隠れ層と出力層を結合する重みを示す。また、$\boldsymbol{b} \in \boldsymbol{R}^{m}$ 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 $\{\boldsymbol{x} \in \boldsymbol{R}^{k\times n}, \boldsymbol{t} \in \boldsymbol{R}^{k\times n'}\}$ 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 $\boldsymbol{H} \equiv G(\boldsymbol{x}\cdot\boldsymbol{\alpha}+\boldsymbol{b}) \in \boldsymbol{R}^{k\times m}$ 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）等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通りである。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 の問題点の一つである局所最適解への収束 [6] 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 は予め全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都度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。

2.2.2 Online Sequential Extreme Learning Machine

本節では OS-ELM について述べる。OS-ELM [2] 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 $\{\boldsymbol{x_i} \in \boldsymbol{R}^{k_i\times n}, \boldsymbol{t_i} \in \boldsymbol{R}^{k_i\times n'}\}$ 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。

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

この時、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。

$$\begin{aligned} \boldsymbol{P_0} &= (\boldsymbol{H_0}\boldsymbol{H_0^T})^{-1} \\ \boldsymbol{\beta_0} &= \boldsymbol{P_0}\boldsymbol{H_0^T}\boldsymbol{t_0} \end{aligned} \tag{5}$$

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良い。

下図は OS-ELM のアルゴリズムの擬似コードであり, 大きく

Algorithm 1 Algorithm of OS-ELM

$\boldsymbol{\alpha} \in \boldsymbol{R}^{n\times m}, \boldsymbol{\beta} \in \boldsymbol{R}^{m\times n'}, \boldsymbol{b} \in \boldsymbol{R}^{m} \Leftarrow random\ input$
$\boldsymbol{H_0} \Leftarrow G(\boldsymbol{x_0} \in \boldsymbol{R}^{k\times n} \cdot \boldsymbol{\alpha} + \boldsymbol{b}) \qquad (k \geqq m)$
$\boldsymbol{P_0} \Leftarrow (\boldsymbol{H_0^T H_0})^{-1}$
$\boldsymbol{\beta_0} \Leftarrow \boldsymbol{P_0 H_0^T t_0}$
$i \Leftarrow 0$
for until $(\boldsymbol{x_{i+1}}, \boldsymbol{t_{i+1}})$ exists **do**
  $\boldsymbol{P_{i+1}} = \boldsymbol{P_i} - \boldsymbol{P_i H_{i+1}^T}(\boldsymbol{I} + \boldsymbol{H_{i+1} P_i H_{i+1}^T})^{-1} \boldsymbol{H_{i+1} P_i}$
  $\boldsymbol{\beta_{i+1}} = \boldsymbol{\beta_i} + \boldsymbol{P_{i+1} H_{i+1}^T}(\boldsymbol{t_{i+1}} - \boldsymbol{H_{i+1}\beta_i})$
  $i \Leftarrow i + 1$
end for

分けて以下に示す 3 つのフェーズに分割される。

（1） **初期化フェーズ**: 重み行列 $\boldsymbol{\alpha}$、$\boldsymbol{\beta}$ の値を任意の確率分布に従う乱数で初期化する。

（2） **初期学習フェーズ**: バッチサイズ k が中間層のノード数 m 以上である $\boldsymbol{x_0} \in \boldsymbol{R}^{k\times n}$ に対する隠れ層の値 $\boldsymbol{H_0}$ を求め、$\boldsymbol{\beta_0}$ とその中間結果である $\boldsymbol{P_0}$ を計算する。

（3） **逐次学習フェーズ**: バッチサイズ k が 1 以上の $\boldsymbol{x_i}$ に対応する隠れ層の値 $\boldsymbol{H_i}$ を求め、$\boldsymbol{\beta_i}$ とその中間結果である $\boldsymbol{P_i}$ を計算する。

初期学習フェーズ、逐次学習フェーズでは逆行列を求める必要があるが、原著ではこれらは全て非正則である場合を考慮し SVD を用いて計算される [2]。また、バッチサイズ k が可変であることにも注目されたい。予め全データが得られている際に、k = 全データ数とすると ELM と同様なバッチ学習になるため、OS-ELM は ELM を一般化したアルゴリズムとして捉えることができる。

2.2.3 OS-ELM の計算ボトルネックの解消

式 4 の計算量的ボトルネックは逆行列演算 $(\boldsymbol{I} + \boldsymbol{H_i P_{i-1} H_i^T})^{-1}$ であるが、$(\boldsymbol{I} + \boldsymbol{H_i P_{i-1} H_i^T})$ の行列サイズは $k \times k$ であるため、$k = 1$ の時、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。

$$\boldsymbol{P_i} = \boldsymbol{P_{i-1}} - \frac{\boldsymbol{P_{i-1} h_i^T h_i P_{i-1}}}{1 + \boldsymbol{h_i P_{i-1} h_i^T}}$$
$$\boldsymbol{\beta_i} = \boldsymbol{\beta_{i-1}} + \boldsymbol{P_i h_i^T}(\boldsymbol{t_i} - \boldsymbol{h_i \beta_{i-1}}) \tag{6}$$

$\boldsymbol{h} \in \boldsymbol{R}^{m}$ 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上に、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。そのため、OSUAD では OS-ELM のバッチサイズを 1 に固定している。

2.2.4 オートエンコーダを用いた異常検知

オートエンコーダ [7] 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一つである。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。この時、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みなせる。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、教師無し学習アルゴリズムに分類される。

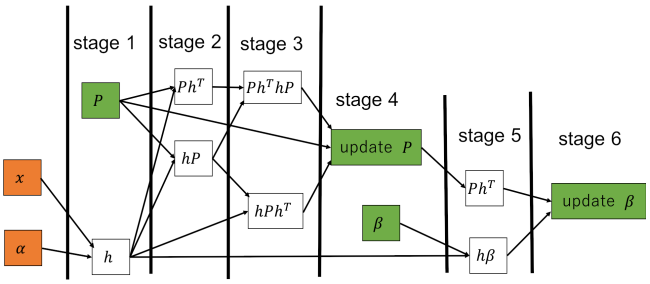

図 2　seq_train の 6 つの計算ステージ

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、教師無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。

3. 設計実装

OSUAD を後述の 4 つの設計オプションで Verilog HDL 実装し、それぞれについて学習器である seq_train モジュールと推論器である predict モジュールについて面積、実行サイクル数をモデル化する。全てのデータを固定小数点として扱い、全体の bit 数を Q、小数部分の bit 数を N とする。オートエンコーダを利用して異常検知を行うため入力層と出力層の数は同じにする必要があり、これを n、隠れ層の数を m とする。固定小数点の加算器、乗算器、除算器の面積、実行サイクル数をそれぞれ S_a、T_a、S_m、T_m、S_d、T_d とする。これらを用いてそれぞれの設計オプションで面積と実行サイクル数のモデル化を行う。除算器は面積は小さいが実行サイクル数の多い bit シフトで行うものと、実行サイクル数は少ないが面積の大きい Newton 法で行うものの 2 種類の設計オプションを検討する。

3.1 パイプライン

式 6 で表される seq_train モジュールの計算を図 2 のように 6 つのステージに分割する。各ステージの間には依存関係が存在し、stage4 で P の値を更新するまでは次の訓練データの stage2 以降に進むことはできない。これを考慮して図 3 のようなパイプラインを構築した。ステージの依存関係を満たすために必要なデータ同士の間隔は T_{stage1}、T_{stage2}、T_{stage3}、T_{stage4} をそれぞれ stage1、stage2、stage3、stage4 の実行にかかるサイクル数とした時、$T_{stage2} + T_{stage3} + T_{stage4} \geqq T_{stage1}$ を満たす時は $T_{stage2} + T_{stage3} + T_{stage4} - T_{stage1}$ となり、満たさない時は 0 となる。このようにパイプラインを構築することで、$max\{T_{stage2} + T_{stage3} + T_{stage4}, T_{stage1}\}$ サイクルに一つの訓練データを学習できる。

3.2 設計オプション

各ステージでは行列同士の演算を行うため、行列の要素同士の掛け算、割り算を行った後に行ごとの総和を求める必要

図 3　seq_train のパイプライン構成 (設計 1、2、3)

がある。今回はツリー型に足し算を行うことにより総和を求めた。その結果、訓練データ 1 回につき seq_train モジュールは $3nm + 4m^2 - 4m - 2$ 回の加算、$4m^2 + 3mn + m$ 回の乗算、m^2 回の除算を行う必要があり、predict モジュールは $2mn + n - m - 1$ 回の加算、$2mn$ 回の乗算を行う必要があった。これらの演算を次の 4 通りの設計オプションで実装した。

- 設計 1:実行サイクル数の最小化
- 設計 2:乗算器の再利用
- 設計 3:乗算器および加算器の再利用
- 設計 4:投機的学習による高スループット化

3.2.1　設計 1:実行サイクル数の最小化

なるべく短い実行サイクル数で演算を終了させるために、全ての演算に一つずつ対応する演算器を使用し、1 つの訓練データごとに各演算器は一度だけ使用する。この設計では seq_train モジュールの演算器部分の面積は、$\left(3nm + 4m^2 - 4m - 2\right) S_a + \left(4m^2 + 3mn + m\right) S_m + m^2 S_d$ とモデル化でき、predict モジュールの演算器部分の面積は $(2mn + n - m - 1)S_a + 2mnS_m$ とモデル化できる。

3.2.2　設計 2:乗算器の再利用

後述する通り、加算器と乗算器を比較した場合乗算器の方が面積は大きい。そのため、面積の制約を考慮して各ステージの乗算器を 1 つの訓練データごとに m 回ずつ利用することで乗算器の個数を減らした。使用する乗算器の数を減らしても、乗算の回数や行う計算自体は変わらないため、精度は変わらない。この設計では 1 つの乗算器を複数回使用するので乗算結果を保持するレジスタが必要となるため、設計 1 よりもレジスタの個数が増えるのでその分面積が大きくなることが考えられる。seq_train モジュールの演算器部分の面積は、$\left(3nm + 4m^2 - 4m - 2\right) S_a + (4m + 3n + 1) S_m + m^2 S_d$ とモデル化でき、predict モジュールの演算器部分の面積は $(2mn + n - m - 1)S_a + 2nS_m$ とモデル化できる。

3.2.3　設計 3:乗算器および加算器の再利用

面積の制約を考慮して、乗算器だけでなく加算器も 1 つの訓練データに対して複数回利用することで演算器の個数を減らす。設計 2 と同様の方法で乗算器の個数を削減するのに加えて、行列同士の乗算の後に生じるツリー型の加算を一つのツリーに対して一つの加算器で行うことにより、使用する加算器の個数を削減した。使用する乗算器、加算器の数を減らすが、乗算、加算の回数や行う計算自体は変わらないため、精度は変わらない。この設計では乗算器だけでなく加算器も複数回使用するため、加算結果を保持するレジスタが必要となるので設計 2 よりもさらにレジスタの個数が増える。よって、設計 2 よりもさらにレジスタの確保で面積が必要となる。seq_train モジュールの演算器部分の面積は、$\left(m^2 + 2n + 4m + 1\right) S_a + (4m + 3n + 1) S_m + m^2 S_d$ とモデル化することができ、predict モジュールの演算器部分の面積は $(2n + m + 1)S_a + 2nS_m$ とモデル化できる。

設計 1、設計 2、設計 3 の seq_train の各ステージの実行サイクル数は表 1 のようにモデル化できる。

表 1　seq_train における各ステージのサイクル数

	設計 1	設計 2	設計 3
T_{stage1}	$T_a + T_m$	$T_a + mT_m$	$(n-1)T_a + mT_m$
T_{stage2}	$T_a + T_m$	$T_a + mT_m$	$(m-1)T_a + mT_m$
T_{stage3}	$T_a + T_m$	$T_a + mT_m$	$(m-1)T_a + mT_m$
T_{stage4}	$T_a + T_d$	$T_a + T_d$	$T_a + T_d$
T_{stage5}	$T_a + T_m$	$T_a + mT_m$	$mT_a + mT_m$
T_{stage6}	$T_a + T_m$	$T_a + mT_m$	$mT_a + mT_m$
合計	$6T_a + 5T_m + T_d$	$6T_a + 5mT_m + T_d$	$(n + 4m - 1)T_a + 5mT_m + T_d$

3.2.4　設計 4:投機的学習による高スループット化

一回の訓練データによって変化する P や β の値は非常に小さい。そのため、前の訓練データで P や β が更新されるのを待たずに次の訓練データの計算に移行しても精度の低下は小さいと考えられる。このように学習を遅延しながら計算を行うことで、高いスループットを実現することができる。最も実行サイクル数の多いステージの稼働率が 100%となるように新しいパイプラインを図 4 のように構築する。表 1 と、オートエンコーダを利用する異常検知では一般的に入出力層数 n よりも隠れ層 m の方が小さいことより、最も実行サイクル数の多いステージは、stage1 もしくは stage4 となる。stage1 の実行サイクル数が多い場合は stage1 が終わり次第、次の訓練データを流し込む。stage4 の実行サイクル数が最も多い場合は、stage1 の計算が終わってから $T_{stage4} - T_{stage1}$ サイクルだけ待ってから次の訓練データを流し込む。このようにパイプラインを変更することで、$max\{T_{stage4}, T_{stage1}\}$ サイクルに一つの訓練データを学習できるようになる。パイプラインを変更した時の設計 1 から設計 3 について 4.5 節でスループットを評価する。

4.　評　　価

4.1　評 価 環 境

設計 1、2、3、4 を Verilog HDL 言語で記述した。RTL シミュレーションには Candece NC-Verilog および SimVision、論理合成には Synopsys Design Compiler、合成ライブラリには Nangate 45nm Open Cell Library を使用した。

4.2　演算器の面積

固定小数点の加算器、乗算器、bit シフトで計算を行う除算器は [8] を利用し、Newton 法を利用して計算を行う除算器は

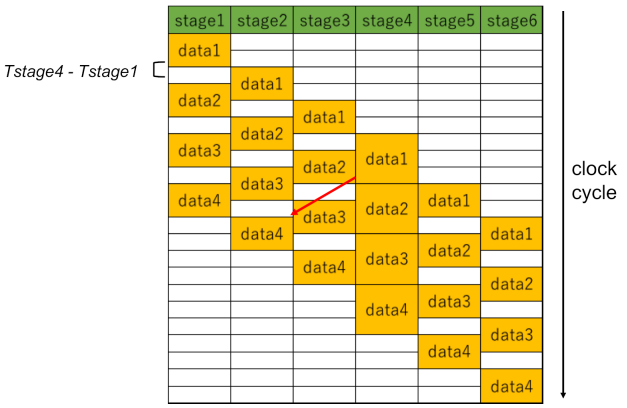

図 4　seq_train のパイプライン構成 (設計 4)

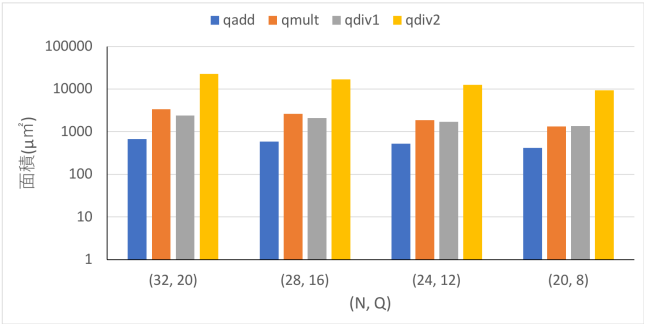

図 5　各演算器の面積

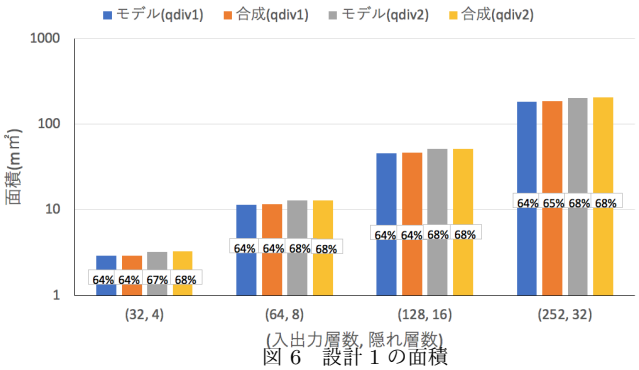

図 6　設計 1 の面積

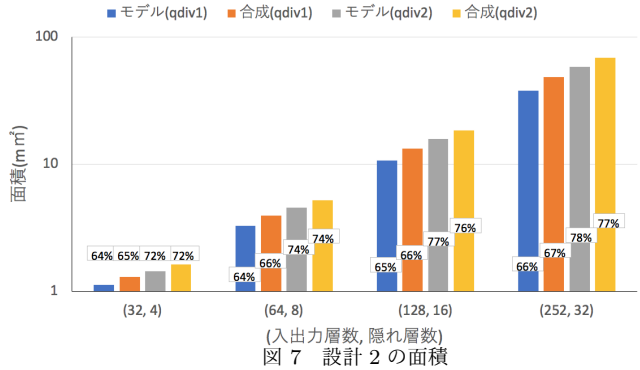

図 7　設計 2 の面積

前述の加算器、乗算器を用いて 1 サイクルで計算が行えるように作成した。各演算器の一つあたりの面積は図 5 のようになった。qadd は固定小数点の加算器、qmult は固定小数点の乗算器、qdiv1 は bit シフトを用いた固定小数点の除算器、qdiv2 は Newton 法を利用した固定小数点の除算器である。小数部分の bit 数を減らし、固定小数の精度を下げることで、1 つ 1 つの演算器の面積を削減できる。小数部分の bit 数を 4bit 減らすごとに面積を qadd は 80〜90%、qmult は 70〜80%、qdiv1 は 80〜90%、qdiv2 は 70〜80%にできた。

seq_train モジュールと predict モジュールの実装には N が 32、Q が 20 のものを使用し、その時の各演算器の面積と実行サイクル数は表 2 のようになった。

表 2　演算器の面積と実行サイクル数

	qadd	qmult	qdiv1	qdiv2
面積 (μ m^2)	679.10	3341.49	2374.32	22453.06
実行サイクル数	1	1	54	1

4.3　面積に関する評価

モデル (qdiv1) は 3. 章でモデル化した式から、qdiv1 を使用した場合の面積を計算したもの、モデル (qdiv2) は qdiv2 を利用する除算器を使用した場合の面積を計算したものとする。合成 (qdiv1) は qdiv1 を用いて実装したモジュールを論理合成することで得た面積であり、合成 (qdiv2) は合成 (qdiv1)、モデル (qdiv1)、モデル (qdiv2) から見積もった qdiv2 を使用した場合の面積である。設計 1、設計 2、設計 3 の seq_train モジュールと predict モジュールの面積の合計はそれぞれ図 6、図 7、図 8 のようになった。図中の数値は seq_train モジュールと predict モジュールの合計の内、seq_train モジュールが占める割合を表している。

モデル (qdiv1) と合成 (qdiv1) について、使用するレジスタの個数が最も少ない設計 1 において差が小さくなり、今回測定したいずれの入力層数、隠れ層数でも 1%程度の差となった。使用するレジスタの個数が最も多い設計 3 がモデル (qdiv1) と合成 (qdiv1) の差が最も大きくなった。入力層 n と隠れ層 m の数が大きくなるにつれて差も大きくなり、n が 32、m が 4 の時は合成 (qdiv1) はモデル (qdiv1) と比較して 44%増加し、n が 256、m が 32 の時は 213%増加した。

今回の実装では除算器の再利用を行っていないため、設計 1、2、3 で合成 (qdiv1) と合成 (qdiv2) の差は同じになった。そのため、全体の面積が小さい場合、合成 (qdiv2) で除算器が占める割合は大きくなり、設計 3 の n が 256、m が 32 の場合では合成 (qdiv2) 全体の面積の内 48%を除算器が占めている。qdiv1 には実行サイクル数が長いが面積が小さいという特徴があり、qdiv2 には実行サイクル数が短いが面積が大きいという特徴がある。そのため、除算器が qdiv1 の場合には今回と同様に除算器の再利用を行わず、除算器が qdiv2 の場合には除算器の再利用を行う設計をすべきであると考えられる。

設計 1、2、3 の合成 (qdiv1) について、n が 256、m が 32 の時、設計 2 は設計 1 の 26%の面積になり、設計 3 は設計 1 の 15%、設計 2 の 56%の面積になった。

4.4　実行時間に関する評価

設計 1、設計 2、設計 3 の seq_train モジュールと predict モジュールの動作周波数を測定した結果、図 9 のようになった。設計 3 は加算結果をレジスタに保持するため、動作周波数は最

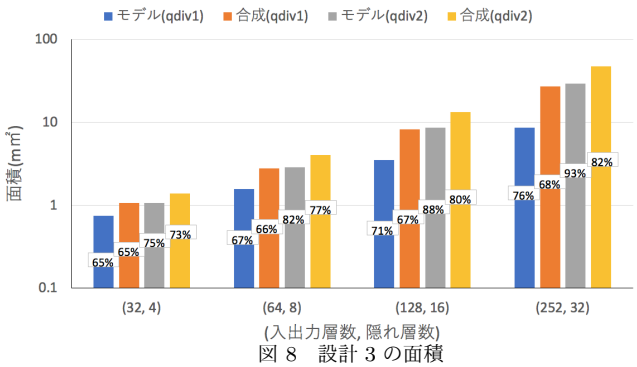

図 8　設計 3 の面積

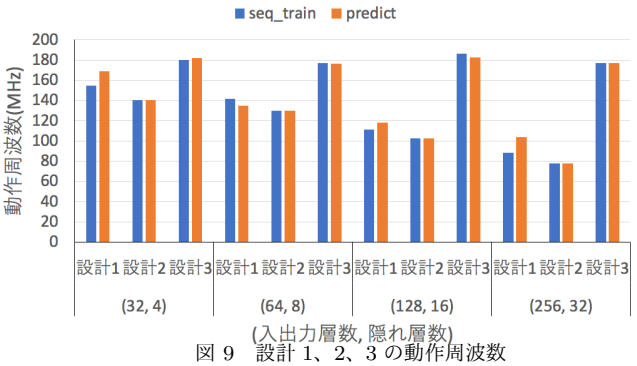

図 9　設計 1、2、3 の動作周波数

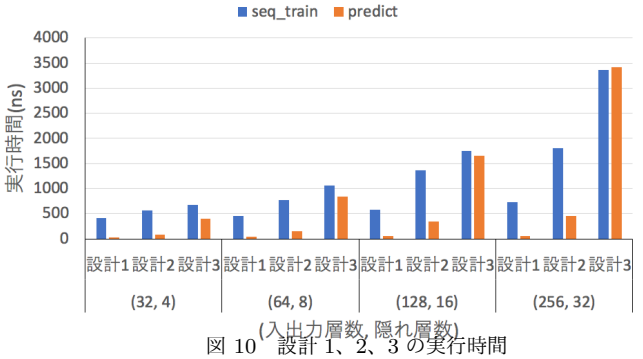

図 10　設計 1、2、3 の実行時間

も高くなった。また、各設計で seq_train モジュールと predict モジュールの間に動作周波数の大きな違いは無かった。

測定した動作周波数と 3. 章でモデル化した実行サイクル数から計算した、1 つのデータに対する seq_train モジュールと predict モジュールの実行時間は図 10 のようになった。設計 3 は動作周波数は最も高くなったが、サイクル数が設計 1 や設計 2 と比較して多いため、実行時間は最も長くなった。また、設計 1 と設計 2 は predict モジュールの実行サイクル数が seq_train モジュールと比較すると非常に少ないため実行時間も短くなったが、設計 3 は predict モジュールの実行サイクル数が多いため、実行時間も長くなった。

4.5　スループットに関する評価

図 3 と図 4 のパイプライン構成について、測定した動作周波数から設計 1、設計 2、設計 3 で 1 秒あたりに学習できる訓練データの個数を表したスループットは図 11 のようになった。T_{stage1} が少ない設計 1 や設計 2、n と m の値が小さい場合の

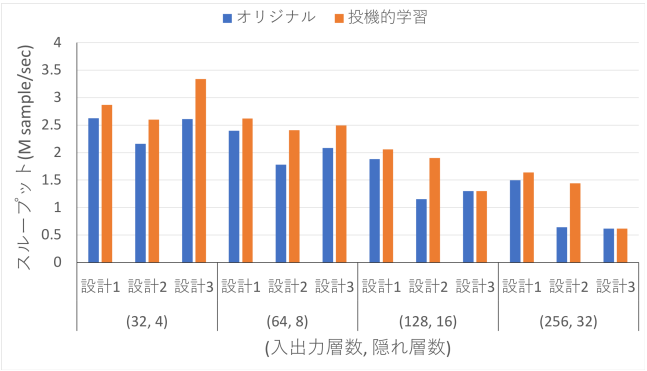

図 11　設計 4 の学習スループット

設計 3 はパイプラインを変更することでスループットを改善できたが、設計 3 は n、m の値が大きくなると T_{stage1} が大きくなるため、スループットを改善できなかった。

5.　まとめと今後の課題

OS-ELM とオートエンコーダを組み合わせた異常検知器である OSUAD を様々な設計方法で実装することで、面積等のコストの制約が厳しいオンデバイス学習器のコストと性能のトレードオフについて評価を行った。

今後の課題として、最適な固定小数の精度を探索することが挙げられる。固定小数の精度を下げることで 1 つ 1 つの演算器の面積を削減することができるため、システム全体の面積を大幅に削減することが期待できるが、固定小数の精度を下げることで異常検知の精度も下がってしまう。そのため、固定小数の bit 幅を変更することによる面積と精度のトレードオフについても明らかにする必要がある。

文　献

[1] M. Tsukada, M. Kondo, and H. Matsutani, “OS-ELM-FPGA: An FPGA-Based Online Sequential Unsupervised Anomaly Detector,” Proceedings of the International European Conference on Parallel and Distributed Computing (Euro-Par’18) Workshops, 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, vol.17, no.6, pp.1411–1423, Nov. 2006.

[3] S. Ravi, “Projectionnet: Learning efficient on-device deep networks using neural projections,” arXiv:1708.00630, Aug. 2017.

[4] A.M. Zyarah, N. Soures, and D. Kudithipudi, “On-device learning in memristor spiking neural networks,” Proceedings of the International Symposium on Circuits and Systems (ISCAS’18), pp.1–5, May 2018.

[5] G.B. Huang, Q.Y. Zhu, and C.K. Siew, “Extreme learning machine: A new learning scheme of feedforward neural networks,” Proceedings of the International Joint Conference on Neural Networks, pp.985–990, July 2004.

[6] G. Marco and T. Alberto, “On the problem of local minima in backpropagation,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.14, no.1, pp.76–86, Jan. 1992.

[7] G. Hinton and R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” Science, vol.313, no.5786, pp.504–507, 2006.

[8] “Fixed point math library for verilog”. `https://opencores.org/projects/verilog_fixed_point_math_library`.