Vol.2018-CSEC-82 No.48
Vol.2018-SPT-29 No.48
2018/7/26

# FPGAインスタンスを用いたクラウドログ異常検知の実装と評価

千田 拓矢[1,a)]　杉尾 信行[1,b)]　青野 博[1,c)]　関野 公彦[1,d)]

**概要**：近年クラウドコンピューティングが普及するとともに，クラウド環境を狙ったサイバー攻撃が多発している．攻撃の原因究明・再発防止のために，セキュリティログを分析することは非常に重要であるが，クラウド環境では日々膨大な種類・量のログが発生するため，従来のような人手による分析は非現実的となっている．そのような課題に対し，機械学習によってログを分析することで攻撃を検知する研究が数多くなされているが，その計算コストの高さからリアルタイムに分析することは難しいのが現状である．本研究ではクラウド環境の FPGA インスタンス上に AutoEncoder による異常検知アルゴリズムを実装し，その性能評価を行う．FPGA を用いることにより，従来の機械学習では難しかった高速でリアルタイムな異常検知を実現させる．

# Implementation and Evaluation of Anomaly Detection on FPGA Instance for Cloud Service Logs

Chida Takuya[1,a)]　Sugio Nobuyuki[1,b)]　Aono Hiroshi[1,c)]　Sekino Kimihiko[1,d)]

## 1. 序論

現在多くの企業や団体がクラウドの導入を進めており，従来のオンプレミスと同等のセキュリティ対策が必要とされている．特に不特定多数の利用者が同じマシンを利用するパブリッククラウドでは，情報漏洩や DDoS 攻撃をはじめとするインシデントが多発しており，安全なクラウド利用に向けた対策は非常に重要な課題である．

そのような対策の一つとして，クラウドサービスのログ分析が挙げられる．クラウドサービスは仮想サーバやストレージなど多くのコンポーネントで構成されているため，取得できるログの量は膨大である．また，近年ではクラウドプロバイダが提供するマネージドサービスや第三者サービスとの連携も進んでおり，取得できるログの種類が複雑化している．今後 IoT デバイスや新たなサービスが発展するにつれて，クラウドのログの肥大化・複雑化はより一層加速すると想定される．

一般的なクラウド環境ではクラウドプロバイダ側でログを収集・分析しているが，それだけでは対処できない攻撃も存在する．例えば，パスワード流出やアクセス制限の設定ミスがあった場合は，クラウドプロバイダから見れば正常なログと見分けがつかない．内部犯による情報漏洩なども同様である．このようなインシデントの場合は，クラウドプロバイダだけでなくクラウド利用者自身が主体的にログを分析・監視する必要がある．

本研究では AutoEncoder を用いたセキュリティログの異常検知を想定し，大量のログを FPGA インスタンスで高速に分析する手法について報告する．ログの分析をいかに高速に行うかという性能面のみに着目しているため，攻撃をどれだけ正確に検出できるかという精度面の評価は行わない．

本論文の構成は以下の通りである．第 2 章では一般的なクラウドに対する脅威と対応するログをまとめ，クラウドログ分析における課題と研究の目的を整理する．第 3 章で

[1] NTT DOCOMO R&D, Yokosuka, Kanagawa, Japan
[a)] takuya.chida.mn@nttdocomo.com
[b)] sugio@nttdocomo.com
[c)] aonoh@nttdocomo.com
[d)] sekino@nttdocomo.com

は事前準備として議論に必要な知識を紹介する．第４章では今回提案する FPGA を用いたログ分析の手法を示し，第５章ではその実装を述べる．第６章でその評価を行い，第７章では結果に対する考察や改善点を述べる．第８章で結論を述べる．

## 2. 課題と研究の目的

本節では，クラウド環境における攻撃とセキュリティログを紹介し，課題と研究の目的について述べる．

### 2.1 クラウドにおけるサイバー攻撃

パブリッククラウド環境での代表的な攻撃例 [2] を下記に示す．

- パスワード/API キーの漏洩
- 仮想サーバやアプリケーションへの不正アクセス
  - デフォルトパスワードの利用
  - 無制限アクセス
- DoS/DDoS などのサービス不能攻撃
- 内部犯行
- その他サービス固有のもの
  - ストレージへのアクセス権限設定ミス
  - 機械学習サービスへの不適切なデータ入力 [3]

クラウド環境では取り扱う鍵やパスワードの種類が多く，適切に管理する必要がある．特に AWS ではアクセスキーを平文のままコードやドキュメントに埋め込んで Web 上に公開してしまうインシデントが頻発しており，第三者によって仮想通貨のマイニングや DDoS 攻撃の踏み台にされてしまう場合がある．また，ファイアウォールで IP 制限をしていない場合や DB や Web サーバなどのアプリケーションを初期設定のまま利用してしまい，第三者に侵入されてしまうというインシデントも発生している．

### 2.2 クラウドにおけるセキュリティログ

パブリッククラウド環境では様々な種類のログを収集・分析することが可能である．攻撃の痕跡や予兆はログに現れるため，ログに対して異常検知を行なうことは非常に重要である．代表的なクラウドサービスのログを**表 1** に示す．

**表 1** クラウド環境で取得できるログ

| 区分 | 種類 |
|---|---|
| アプリケーション | アクセスログ |
| | OS ログ (syslog など) |
| | IDS/IPS ログ |
| ミドルウェア | トラフィックログ |
| | API ログ |
| ハードウェア | マシンログ（サーバ，デバイス） |
| | ※ CPU,Disk の使用率など |

ほとんどのログはクラウドプロバイダから利用者に提供されているが，データセンターのサーバ基盤のマシンログやトラフィックログなど利用者に提供されないログも存在する．反対に仮想マシンの OS ログや IoT デバイスのログなどクラウド利用者が自己責任で管理するログもある．そのようなログはクラウドプロバイダの責任範囲外である[*1]ため，クラウド利用者自身でログを収集・分析する必要がある．

特に近年では様々な用途で IoT デバイスが実用化され，IoT デバイスを狙った攻撃も急増している [1]．多くの場合，IoT デバイスからの大量な情報を処理するためにクラウドサービスにデータを転送することが多いため，クラウド環境上で IoT デバイスのログの収集・管理を行うことが可能である．

このようにクラウドにおけるログの量は今後ますます増加していくと予想され，クラウド利用者は複雑で肥大化するログを分析していかなければならない．

### 2.3 クラウドログの異常検知における課題

**肥大化するログに対する異常検知**

クラウド環境では様々な攻撃が存在し，利用者が見るべきセキュリティログも増加している．今後，クラウド利用者やデバイスの数が増加するにつれて，分析すべきログの量も肥大化していくことが予想されるため，人間の目による監視は非現実的なものとなっている．

この問題を解決するために，機械学習を利用してセキュリティログの異常検知を行う研究が進んでいる．特に深層学習を用いた研究では，人間よりも高精度な判別が実現しており，多くの分野で導入されている．その中でも AutoEncoder を用いた異常検知 [5] は，柔軟に仕様を変更することが可能であり，高精度でロバストな異常検知手法として期待がかかっている．このように機械学習を利用してログの監視を自動化することで，人手による負担を軽減できるだけでなく，より高精度な分析が可能となる．

**リアルタイムな異常検知**

インシデントを早期発見・防止するためには，リアルタイムにログを取得して異常検知することが重要であるが，機械学習の計算量は分析対象のデータサイズやパラメータ数に比例することが多く，高精度なアルゴリズムを適用した場合は非常に時間がかかってしまう恐れがある．

一般的には機械学習の計算時間を削減するために，GPU や ASIC などの特殊なハードウェアを利用することが多い．特にクラウド環境では手元にハードウェアを用意する必要がないため，高機能な CPU や GPU 搭載のインスタンスを簡単に利用することができる．最近ではそのような特殊なハードウェアの一つとして FPGA 搭載のインスタンス

[*1] https://aws.amazon.com/compliance/shared-responsibility-model/?nc1=h_ls

Vol.2018-CSEC-82 No.48
Vol.2018-SPT-29 No.48
2018/7/26

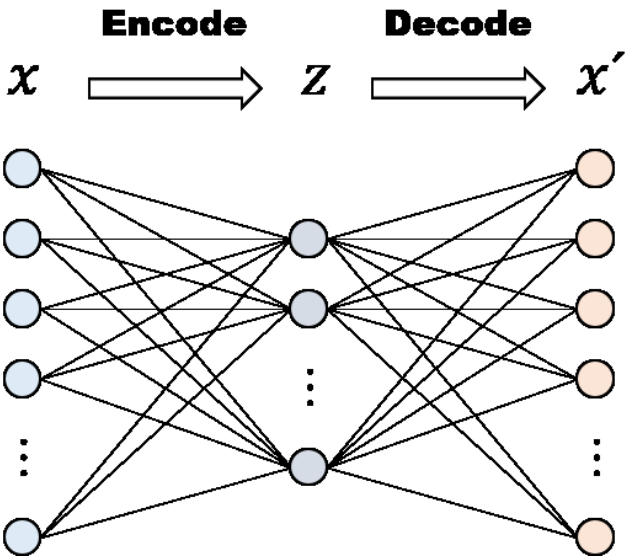


図 1　Auto Encoder の概観図

も提供されている．FPGA インスタンスでは利用者自身がハードウェアの回路設計を行うため，演算のボトルネックとなる部分を高速化することが出来る．また利用料金も比較的安価であるため，セキュリティ用途のような 24 時間常に異常検知を行う場合に有力な手段だと考えられる．

### 2.4　本研究の目的

クラウドログの異常検知では自動化とリアルタイム性を両立させる必要がある．本研究では，機械学習アルゴリズムとして **AutoEncoder** を採用し，高速化のために **AWS FPGA インスタンス**を導入した異常検知システムを提案する．また，システム実現に向けて，FPGA を利用した AutoEncoder による異常検知を実装し，推論速度の評価を行う．

## 3.　事前準備

本節では事前準備として AutoEncoder による異常検知と FPGA インスタンスの概要，カーネルプログラムの最適化について説明する．

### 3.1　AutoEncoder による異常検知

AutoEncoder のネットワーク構成図を**図 1** に示す．

AutoEncoder は基本的な多層ニューラルネットと同様に複数のレイヤで構成され，入力データと出力データの誤差（再構成誤差）が小さくなるように学習を行う．入力層から中間層までの変換を Encode，中間層から出力層までの変換を Decode と呼ぶ．AutoEncoder の学習と推論は独立した処理であるため，ネットワークの構成が同じであれば，事前に複数のパラメータを用意しておくことが可能である．

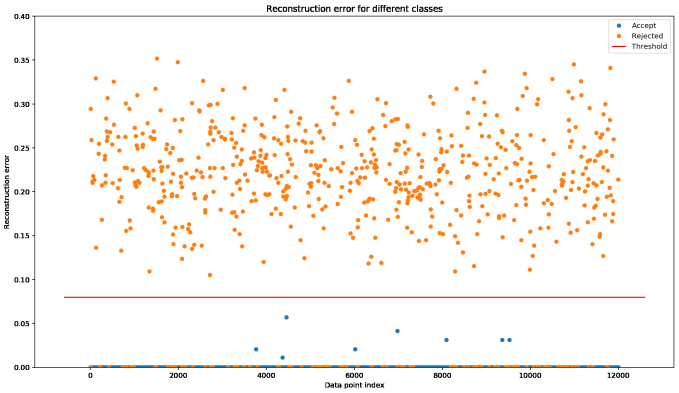


図 2　VPC Flow Logs の再構成誤差

AutoEncoder による異常検知の手順[*2]を下記に示す．

- AutoEncoder の学習
  - (1) AutoEncoder のレイヤの種類，パラメータ数などを決める．
  - (2) 正常な期間のデータを用意して AutoEncoder を学習させる．
  - (3) 再構成誤差を計算し，異常度を判定する閾値を決定する．
- AutoEncoder の推論
  - (1) 分析対象となるデータを AutoEncoder へ入力する．
  - (2) 再構成誤差を計算し，閾値と比較する．

**図 2** は AWS で取得したトラフィックログを対象に異常検知を行った例である．正常・異常が事前にラベル付けされていれば，通常の二値分類と同様に Accuracy や Recall などの指標を元に閾値を決定することが可能である．ただし，実用上は正常・異常のラベル付けは困難であることが多いため，その場合は攻撃がなかった期間のログを正常と見做して学習させて，再構成誤差が全て閾値内に収まるように閾値を決定する．

### 3.2　AWS FPGA インスタンスの開発フロー

FPGA インスタンスの開発環境を**図 3** に示す．

開発者はビルド用の開発インスタンスとアプリケーション実行用の FPGA インスタンスの二種類を用意する．開発インスタンスを別に用意するのはビルド時間を短縮するためであり，高性能な CPU やメモリを搭載したインスタンスが推奨されている．インスタンスは AWS が用意した FPGA Developer AMI から作成する．

AWS で FPGA インスタンスを利用するために，開発者は AFI と呼ばれる回路イメージを作成する必要がある．AFI とは FPGA インスタンス上で FPGA デバイスに回路を展開するために必要なイメージファイルである．AFI は

[*2] https://github.com/curiousily/Credit-Card-Fraud-Detection-using-Autoencoders-in-Keras/blob/master/fraud_detection.ipynb

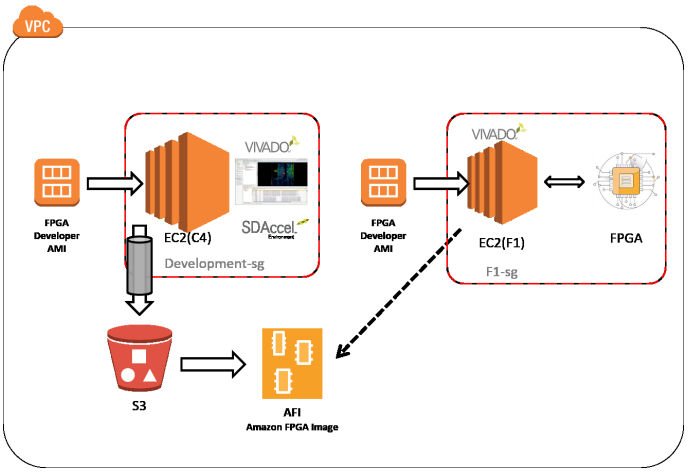


図 3　FPGA インスタンスの開発環境

AWS 側で管理されており，第三者へ公開したり，AFI を購入して利用することも可能である．

AWS FPGA インスタンスの開発には Xilinx 社の SDAccel 開発環境 [4] を利用する．SDAccel はクラウドデータセンター向けの CPU-FPGA 開発環境であり，C/C++などの高級言語でハードウェア開発が可能である．SDAccel 開発環境では実際の AFI 合成の前にエミュレーションによって処理性能を検証することができる．

実際の開発フローをまとめる．まず第一に開発者はビルド用のインスタンスで回路設計・合成を行い，出力されたバイナリファイルを S3[*3]へ転送する．次に S3 へ転送したファイルを AFI へ変換するように AWS へコマンドを送る．AWS 側で変換が終わると FPGA インスタンスから AFI を読み込んで利用することができる．

### 3.3　AWS FPGA インスタンスのプログラム実行フロー

FPGA インスタンスでの処理は下記の通り 2 つのプログラムから構成される．

- ホストプログラム：FPGA インスタンスのホスト (CPU) で実行するプログラム
  - デバイス情報の管理
  - 入力データの前処理
  - デバイスへのデータ入出力
  - 出力データの後処理
- カーネルプログラム：FPGA インスタンスのデバイス（FPGA）で実行するプログラム
  - ホストへのデータ入出力
  - 演算処理
  - 他のデバイスへのデータ入出力

ホストプログラムでは FPGA デバイスの情報取得やカーネルプログラムの実行に必要なデータの入出力を行い，カーネルプログラムでは高速化したい演算処理を実行する．実装言語として C/C++，OpenCL，HDL（ハードウェア記述言語）などが利用できる．実際にハードウェアの回路として合成されるのはカーネルプログラムであり，ホストプログラムは通常のプログラムと同様に実行ファイルとして生成される．

[*3] https://aws.amazon.com/jp/s3/

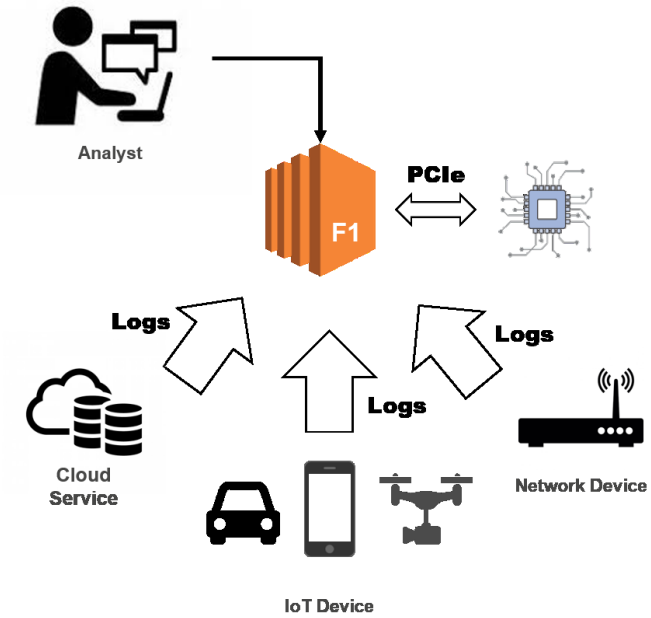


図 4　FPGA インスタンスを用いたリアルタイムログ分析システム

### 3.4　カーネルプログラムの最適化

本研究ではカーネルプログラムの処理を高速化するためにループアンローリングとループパイプラインという手法を用いる．SDAccel 開発環境ではループパイプラインのために xcl_pipeline_loop，ループアンローリングのために opencl_unroll_hint という最適化オプションが用意されている．本研究の実装ではこれらのオプションを用いてカーネルプログラムの最適化を行う．

ループアンローリングではプログラム内の繰り返し処理に対して，複数の計算資源を用意する．これによってループ一回で実行する演算が一回ではなく複数回に展開されるためプログラムのスループット，レイテンシが向上する．ループパイプラインではループ内の処理を複数ステージに分割して，それぞれのステージに対して計算資源を用意する．パイプライン化によって各ループの実行で前のループの実行完了を待たずに次のループの実行を開始することが可能となる．

## 4.　提案システム

本節ではクラウド環境において大量のログをリアルタイムに分析するために，FPGA インスタンスを用いた AutoEncoder による異常検知システムを提案する．

### 4.1 概要

提案システムの俯瞰図を図 4 に示す．提案システムではクラウド環境で発生した API ログやトラフィックログ，IoT デバイスから収集したログ，ファイアウォールや IPS/IDS などのネットワーク機器から収集したログなどを対象とし，これらのログを FPGA インスタンスに集約して異常検知を行う．異常検知には AutoEncoder を用いるため，転送されてきたログが到着すると順次分析を開始し，リアルタイムに異常検知を行うことが可能である．ログを分析するセキュリティアナリストは FPGA インスタンスに接続して，可視化された異常検知の結果をブラウザ経由で確認する．

### 4.2 パラメータの学習と転送

ニューラルネットの学習と推論は独立しているため，学習には GPU インスタンス，推論には FPGA インスタンスを用いることができる．AutoEncoder の学習は Tensorflow や MXnet といった既存のフレームワークによって行い，重みやバイアスなどのパラメータのみ取り出す．取り出したパラメータは事前に FPGA インスタンス上に転送しておき，推論処理の実行開始の際に FPGA デバイス上へパラメータを転送する．

提案システムでは，ニューラルネットの構成が変わらなければパラメータはいつでも更新可能である．ニューラルネットの構成が変わる場合はそのモデルごとに AFI を作成することで対応する．FPGA インスタンスは AFI の切り替えも可能であるため，パラメータだけでなくモデルの変更にも柔軟に対応することができる．

### 4.3 FPGA インスタンス上での異常検知

AutoEncoder による異常検知は FPGA インスタンスで行う．第 3 節で説明した通り，FPGA インスタンスでは CPU で実行されるホストプログラムと，FPGA 上で実行されるカーネルプログラムの二種類が存在する．

提案システムでは，ホストプログラムでは取得したログの前処理を行い，FPGA デバイスへ入力データやパラメータを転送する．カーネルプログラムではホスト側から受け取ったログやデータを FPGA デバイス上のメモリへコピーし，AutoEncoder による推論処理を行う．

提案システムでは再構成誤差の計算も FPGA 上で実行する．その結果，FPGA からホスト側に出力するデータは異常度を表す再構成誤差のみとなり，入力データと比べて転送すべきデータ量を削減することが可能である．

### 4.4 提案システムの利点

提案システムの利点としてコストパフォーマンスとカスタマイズの柔軟性の二つが挙げられる．

従来では AutoEncoder の推論を実行するプロセッサとして CPU もしくは GPU が主流であるが，第 2 節で述べたように，CPU による推論処理ではリアルタイムな異常検知は困難になると予想され，GPU による推論処理では高コストである．それに対して，提案システムでは新たに FPGA を導入することで，CPU よりも高速でありながら GPU よりも低コストな推論処理を実現させる．

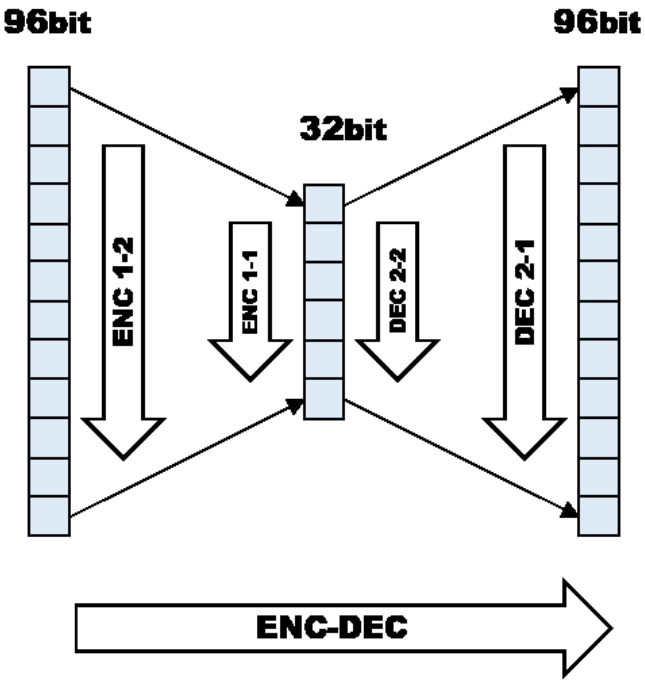


図 5 AutoEncoder の各層とループの対応

FPGA インスタンスではパラメータや FPGA イメージの再読み込みが可能であるため，AutoEncoder のカスタマイズにも柔軟に対応することが可能である．これは FPGA の再構成可能という特性によるものである．提案システムではそのような柔軟性を活かし，ログの傾向の変化や新規ユーザやデバイスの追加にも柔軟に対応する．定期的に再学習を行なってパラメータを最新に保つことや，対象となるログごとに AutoEncoder のモデルを切り替えることが可能である．

## 5. 実装

本節では実装の設計と実装方法について述べる．

### 5.1 分析対象とするログ

本研究では AWS の VPC Flow Logs[*4] を対象に異常検知の検証を行う．VPC Flow Logs はクラウド上の仮想ネットワークへのアクセスに関するトラフィックログであり，クラウド利用者は自由に取得可能である．

トラフィックログから下記の四つの情報を抽出し，96 個の 1/0 のビット列へ変換する．これを AutoEncoder の入力データとする．また，VPC Flow Logs にはアクセス制

[*4] https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/flow-logs.html

限に基づいて Accept/Reject の情報を含んでおり，これを正解ラベルとしている．

- source IP address : 32bit
- source Port : 16bit
- destination IP address : 32bit
- destination Port : 16bit

### 5.2 AutoEncoder の設計

本研究では三層の AutoEncoder を FPGA 上に実装した．実装は第 3 節の開発フローに沿っている．FPGA インスタンスにはリアルタイムにログが転送され，定期的に一定サイズのログを分析するものとする．

入力層は 96 個のビット列をそのまま用いる．中間層のノード数は 32，出力層は 96 とする．各層は全て全結合層である．活性化関数には標準シグモイド関数を用いた．

実装した AutoEncoder の擬似コードを Algorithm1 に示す．擬似コードのループを図示したものが**図 5** である．

**Algorithm 1** 3-Layer AutoEncoder

```
for i < MAX_LOGS do  //ENC-DEC
    for j < N_M do  //ENC 1-1
        for k < N_I do  //ENC 1-2
            ...
        end for
    end for
    for j < N_O do  //DEC 2-1
        for k < N_M do  //DEC 2-2
            ...
        end for
    end for
end for
```

$N_I$・$N_M$・$N_O$ はそれぞれ，入力層・中間層・出力層のノード数を表しており，$MAX_{LOGS}$ は一度に読み込むログのサイズである．

擬似コードの ENC-DEC はログを一行ずつ読み込んで AutoEncoder へ入力するためのループである．ENC 1-1/ENC 1-2 は AutoEncoder の Encode 部分の全結合のニューラルネットの演算を行い，DEC 2-1/DEC 2-2 は Decode 部分の演算を行う．

### 5.3 AutoEncoder の実装

**事前準備**

AutoEncoder のパラメータを得るために，同一の AutoEncoder を Python3.6 を用いて実装し，GPU インスタンス上で学習を行った．フレームワークは Keras と Tensorflow を用いた．学習が完了したモデルのパラメータは FPGA インスタンスのストレージに転送した．

**FPGA での推論処理プログラム**

第 3 節で示した通り，ホストプログラムとカーネルプログラムを実装した．実装には C++と OpenCL を用いている．ホストプログラムでは主にログファイルの読み込みや FPGA へのデータ転送などを行い，実際の AutoEncoder の推論処理は FPGA 上で実行される．推論結果はホストプログラムへ出力され，閾値との比較はホストプログラムで行う．

**CPU での推論処理プログラム**

CPU での性能評価のために擬似コードで示したものと同一の構造のプログラムを C++によって実装した．AutoEncoder の推論処理の実行時間を計測出来るように，擬似コードの部分を関数として切り出した．

## 6. 評価

本節では CPU と FPGA の性能評価について，その実験方法と結果を述べる．

### 6.1 実験概要

本研究では FPGA インスタンスに搭載されている CPU と FPGA を対象に AutoEncoder の推論速度の比較を行う．

まず，AutoEncoder のループへ与える最適化オプションの違いによって，性能がどの程度変化するかをエミュレーション上で評価する．その後 FPGA インスタンスの実機上で AutoEncoder の推論処理を実行し，異常検知に必要な時間を CPU と FPGA で比較する．

実験に用いた FPGA インスタンス (f1.2xlarge) の環境を**表 2** に示す．

**表 2** 実験環境

| | |
|---|---|
| OS | CentOS 7.4 |
| コンパイラ | g++ (GCC) 4.8.5 20150623 (Red Hat 4.8.5-16) |
| CPU | Intel Xeon E5-2686 v4 (vCPU = 8) |
| FPGA | Xilinx Virtex UltraScale+ VU9P |

### 6.2 評価モデル

AutoEncoder の推論速度の性能比較のために下記の 6 つのモデルを用意した．FPGA での推論モデルは第 3 節で述べた最適化オプションの組み合わせを変化させたものである．また，CPU での推論モデルはコンパイラの最適化オプションを変化させたものである．各モデルを**表 3** に示す．

**表 3** 実験に用いるモデル一覧

| ハードウェア | モデル名 | 最適化オプション |
|---|---|---|
| FPGA | Default | なし |
| | Pipeline | xcl_pipeline_loop |
| | Unroll | opencl_unroll_hint |
| | All | xcl_pipeline_loop<br>opencl_unroll_hint |
| CPU | O0（最適化なし） | g++ -O0 |
| | O2（最適化あり） | g++ -O2 |

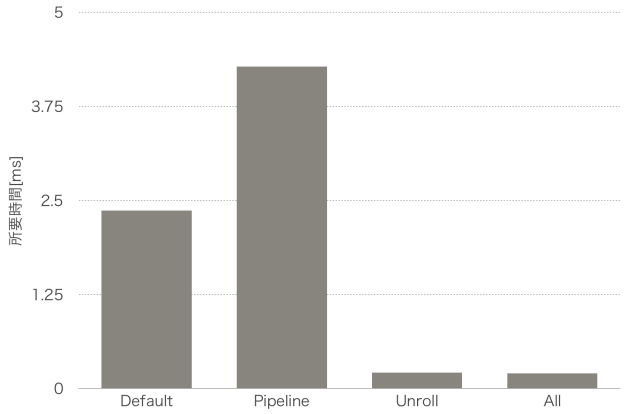


図 6　エミュレーションによる性能評価

これらのモデルでログの異常検知処理を実行し，AutoEncoder の推論処理の部分に費やした時間を計測する．実機検証では読み込ませるログを 16 行，128 行，256 行，512 行，1024 行と変化させて評価する．

### 6.3　エミュレーションでの性能評価

カーネルプログラムの挙動をエミュレーションした結果を図 **6** に示す．エミュレーションでは 16 行のログ（約 1.6KB）に対して異常検知を行なった場合の処理時間を計測している．結果を見ると Unroll モデルは処理時間が短く，ループアンローリングが有効に働いていることがわかる．

逆に，Pipeline モデルは何も最適化オプションを付けない Default モデルよりも性能が低く出ている．これは今回実装に用いた開発環境では最適化オプションを何もつけなかった場合，自動的にパイプラインの最適化が働くためである．

### 6.4　実機での性能評価

FPGA インスタンス実機上での性能評価の結果を図 **7** に示す．Default モデルと Pipeline モデルは除外している．Opt モデルは All モデルを拡張して，AutoEncoder の推論処理以外のメモリコピーや再構成誤差の計算まで最適化したモデルである．

結果を見ると CPU のコンパイルに最適化オプションをつけていない場合は FPGA の方が高速である．しかし，CPU のコンパイルに最適化オプションをつけた場合は CPU の方が高速である．また，FPGA のモデルの実機での所要時間とエミュレーションによる所要時間はほぼ同じであった．FPGA のモデル同士の比較では Opt モデルがもっとも高速である．

## 7.　考察

本節では性能評価の結果に対する考察を行い，改善点や今後の課題について述べる．

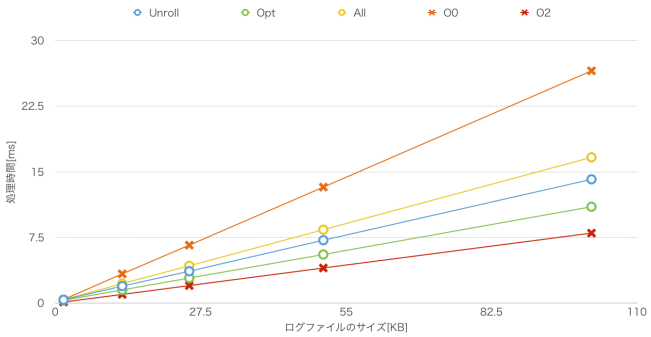


図 7　各モデルの性能比較の結果

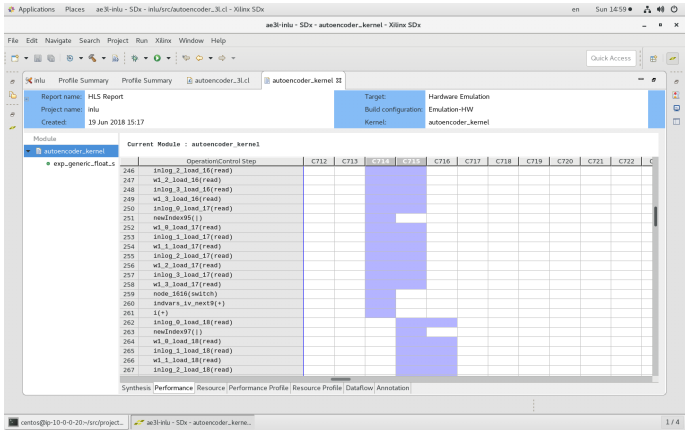

図 8　ループアンローリングによる並列実行の様子

### 7.1　推論処理のボトルネック

ニューラルネットを構成する一つのニューロンは下記の計算式でモデル化される．

$$z_j = f(\sum_{k=1}^{n} w_k x_k + b_j)$$

本研究ではループアンローリングにより重みと入力値の積を求める演算処理（$w_k x_k$）は並列に求めることが出来た．図 **8** ではループアンローリングによって ENC1-1 が並列実行される様子が確認できる．一方，積の累積和を求める演算（$\sum_{k=1}^{n}$）は並列に処理することができず，ボトルネックとなっていることがわかった．図 **9** では演算処理が並列に実行できず，独立に行われていることがわかる．このような配列の総和を求める処理は Reduction と呼ばれており，部分和を求める方法やメモリ配置を変更することでさらなる高速化が見込める[*5]．

### 7.2　AutoEncoder の拡張性

異常検知の精度向上のために AutoEncoder のパラメータの数や層数を変更する必要がある．今回実装した三層

[*5] http://www.fz-juelich.de/SharedDocs/Downloads/IAS/JSC/EN/slides/opencl/opencl-05-reduction.pdf

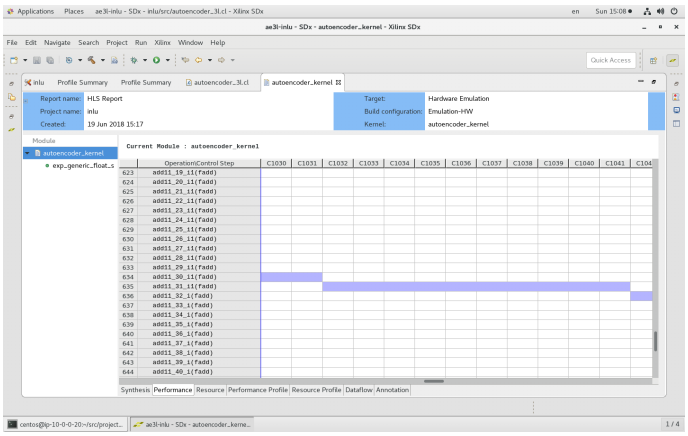

図 9　ボトルネックとなっている累積和の演算処理

の AutoEncoder では FPGA デバイスの回路面積の 2%程度しか利用していないため，拡張の余地は十分である．また，全結合層ではなく畳み込み層や正規化層などの導入も可能である．畳み込み層の演算処理は既存研究 [6] によって FPGA で高速化できることが示されている．

### 7.3　ニューラルネットのデータ型の低精度化

本研究の実装のデータ型には全て 32bit の float 型を用いているが，より低精度な 16bit の float 型や 8bit の int 型を導入することでさらなる処理時間の削減が見込める．Deephi 社の研究 [7] によると低精度なデータ型を用いてもニューラルネットの精度低下はごく僅かであるということが判明している．特に異常検知のようなクラス数の少ない分類では，低精度化の影響は比較的少ないと考えられる．さらに進んだ高速化手法としてニューラルネットのバイナリ化の研究も進んでいる [8], [9]．

### 7.4　リアルタイム処理に向けた考察

実機検証の結果，Opt モデルで 1024 行のログファイルに対する異常検知の所要時間は 11.013[ms] であった．所要時間とログのデータサイズはほぼ線形に比例しているため，本研究の実装では一秒間に 9000 行以上のログを処理することができると推測される．想定システムの規模やログの種類によってリアルタイム性の要件が異なるため，提案システムが妥当かどうかの判断はここでは難しい．

また，ログの前処理や AutoEncoder のパラメータ数によって処理に必要な時間も異なってくるため，リアルタイム性と異常検知の精度を両立させるような AutoEncoder のモデルを構築していく必要がある．

## 8.　結論

本研究では FPGA インスタンス上に AutoEncoder による異常検知システムを実装し，その性能評価を行なった．トラフィックログの異常検知を対象に CPU と FPGA で処理速度の比較を実施し，FPGA による推論処理は高速でかつ拡張性が高く，リアルタイム処理に向けて有力な候補の一つであることを確認した．

今後の課題として AutoEncoder の異常検知の精度を評価すること，ログの種類に応じてリアルタイム処理に必要な要件を整理すること，ハードウェアの実装を最適化してさらなる高速化を目指すことが挙げられる．特にハードウェア実装に関しては課題が多く，今回の実装ではデータ型の最適化やボトルネックの解消が不十分であり FPGA インスタンスの性能を完全に引き出せたとは言い難い．ニューラルネットのバイナリ化やパラメータ数の変更など，今後に向けてさらなる改善の余地が残されている．

## 参考文献

[1] ”NICTER 観測レポート 2017” 国立研究開発法人 情報通信研究機構サイバーセキュリティ研究所 サイバーセキュリティ研究室
[2] ”Treacherous 12 - Top Threats to Cloud Computing + Industry Insights” Cloud Security Alliance.
[3] Yuan, Xiaoyong, et al. ”Adversarial Examples: Attacks and Defenses for Deep Learning.” arXiv preprint arXiv:1712.07107 (2017).
[4] Xilinx. ”UG1023 SDAccel Environment User Guide (ver2018.2)”
[5] Zhou, Chong, and Randy C. Paffenroth. ”Anomaly detection with robust deep autoencoders.” Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2017.
[6] Suda, Naveen, et al. ”Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks.” Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2016.
[7] Guo, Kaiyuan, et al. ”Software-Hardware Codesign for Efficient Neural Network Acceleration.” IEEE Micro 37.2 (2017): 18-25.
[8] Umuroglu, Yaman, et al. ”Finn: A framework for fast, scalable binarized neural network inference.” Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2017.
[9] Nurvitadhi, Eriko, et al. ”Accelerating binarized neural networks: comparison of FPGA, CPU, GPU, and ASIC.” Field-Programmable Technology (FPT), 2016 International Conference on. IEEE, 2016.