APIコールグラフを用いたマルウェアの検知における APIコールの引数を考慮するように拡張した Extended GCNの性能の評価

荒井 康汰[1,a)]　上原 哲太郎[2]

概要：高度化，および多様化するマルウェアへ対策するためには，人工知能の活用が不可欠となっている．従来手法として API コールグラフを用いた GCN によるマルウェアの振る舞い検知がある．この手法では，API コールグラフの頂点に API コールに関する特徴量を割り当て，その特徴量を用いてグラフ畳み込み演算を行う．この制約により，GCN は API コールの引数を考慮することができない．そこで，本研究では API コールを用いたマルウェアの振る舞い検知において引数を考慮できるように拡張した Extended GCN を提案する．EGCN は，頂点の隣接関係に基づいて頂点と辺の特徴量を結合させてから畳み込み演算を行う．有効性を評価するため，EGCN，GCN，LSTM, GRU, RNN を実装してマルウェアを判別させる実験を行った．その結果，EGCN が LSTM や GRU よりも優れた性能を発揮することは示されなかったが，学習済みパラメータからマルウェアの判別において重要でない API，および引数が存在することは示された．

キーワード：グラフ畳み込みネットワーク，マルウェア，API コール，引数，深層学習

Performance Evaluation of Extended GCN to consider API Call Arguments in Malware Detection with API Call Graphs

Kouta Arai[1,a)]　Tetsutaro Uehara[2]

Abstract: Using Artificial Intelligence has become essential to address advanced and massive malware. To this end, malware behavior detection with API call graphs using GCN was proposed. In this method, the features of API calls are assigned to the graph vertices, and Graph Convolutional operation uses these features. However, GCN cannot consider API call arguments because GCN uses only the features of the graph vertices. In this paper, we propose Extended GCN (EGCN) that can consider API call arguments for malware behavior detection. Before operating graph convolution, ECGN concatenates the features of vertices and edges based on the adjacency of vertices and convolute with these features. To evaluate the effectiveness of EGCN, we implemented EGCN, GCN, LSTM, GRU, and RNN. Experimental shows that EGCN has inferior performance than LSTM and GRU. However, the learned parameters of EGCN show that many APIs and arguments are not important for malware detection.

Keywords: Graph Convolutional Network, malware, API call, arguments, Deep Learning

1. はじめに

マルウェアをはじめとするサイバー攻撃は増加し続けており，過去 1 年間で新種や亜種のマルウェアが 1.6 億個以上も発見されている [1]．特に，標的型攻撃で用いられるマ

[1] 立命館大学大学院情報理工学研究科
Graduate School of Information Science and Engineering, Ritsumeikan University
[2] 立命館大学情報理工学部
College of Information Science and Engineering, Ritsumeikan University
[a)] arai@cysec.cs.ritsumei.ac.jp

ルウェアは，セキュリティ製品の検知を回避するために，標的のシステムを入念に調査したうえで開発されている．実際，標的型攻撃を受けた三菱電機や日本電気は，攻撃を検知できなかったと報告している [2], [3]．このように，人手だけでマルウェアを対策するのは限界であり，人工知能を用いて対策を代替，あるいは補助する必要性が高まっている．

近年，人工知能の一分野である深層学習は，画像認識や自然言語処理といった様々な分野で人間を超える性能を達成している [4], [5], [6], [7]．そのため，深層学習の手法である CNN や RNN をマルウェアの検知に応用する研究は多い．しかし，それらの手法はプログラムが持つ分岐処理や反復処理を明示的に考慮することができない．この欠点を克服するために，グラフニューラルネットワーク（GNN）によるマルウェアの判別も研究されている．

GNN はグラフの構造に基づいて演算を行うため，入力となるグラフが必要である．グラフを用いてマルウェアを判別する研究では，命令グラフ，制御フローグラフ，API コールグラフのいずれかが主に使用される．また，グラフを作成するには，静的解析か動的解析のいずれかを行ってマルウェアから特徴を抽出する必要がある．静的解析は，プログラムのコードを読み解く手法であるため，理論的にはいずれのグラフも作成することができる．しかし，マルウェアのコードは難読化されていることが多いため，実際には静的解析からグラフを作成することは困難である．一方，動的解析は，プログラムを実行させ，その挙動を監視する手法であるため，難読化に対して堅牢である．また，この監視によって API コール列が収集されるため，API コールグラフを作成することができる．

GNN を API コールグラフに適用してマルウェアを判別する研究では，Kipf らが提案したグラフ畳み込みネットワーク（GCN）[8] がよく用いられる [9], [10]．しかし，この GCN はグラフの頂点に割り当てられた特徴量のみを用いてグラフ畳み込み演算を行う．すなわち，GCN は API の引数を考慮することができない．この課題を解決するため，本研究では，API コールグラフを用いたマルウェアの振る舞い検知において引数を考慮できるように拡張した Extended GCN (EGCN) を提案する．EGCN は頂点の隣接関係に基づいて頂点と辺の特徴量を結合させてからグラフ畳み込み演算を行う．

提案手法の有用性を評価するため, EGCN, GCN, LSTM, GRU, RNN にマルウェアか否かを判別させる実験を行う．

本論文の構成は以下の通りである．第 2 章では，研究背景について述べる．第 3 章では，提案手法を説明する．第 4 章では，マルウェアを判別させる実験を行い，性能を評価する．第 5 章では，結果に対して考察する．第 6 章では，本論文のまとめを述べる．

2. 研究背景

本章では，API コールグラフ，および GCN の定義，関連研究を示す．

2.1 API コールグラフ

多重グラフ G を $G = (V, E)\,(V \subseteq \mathbb{N})$ とする．このとき，V, E は G の頂点，および有向辺の集合である．また，API と $i \in V$ を 1 対 1 に対応させることで，辺 $e_{ij_m} \in E$ は i から j という順序で API が呼び出されたことを表現できる．また，多重辺を区別するために m を用いる．

2.2 グラフ畳み込みネットワーク（GCN）

GCN は，Kipf らが提案した GNN の実装の一種であり [8]，下式で定義される．なお，説明の都合で添字を追加している．

$$\boldsymbol{h}_i^{(l+1)} = \boldsymbol{f}^{(l)}\left(\boldsymbol{b}^{(l)} + \sum_{(j,m)\in\mathcal{N}_{in}(i)} \frac{1}{c_{ji}^{(l)}} \boldsymbol{h}_j^{(l)} W^{(l)}\right) \quad (1)$$

ここで，$\boldsymbol{h}_j^{(l)}$，$f^{(l)}(\cdot)$，$\boldsymbol{b}^{(l)}$，$W^{(l)}$，$c_{ji}^{(l)}$ はそれぞれ l 層目の，隠れ層の頂点 j の出力，活性化関数，バイアス，パラメータ，辺 e_{ji_m} に関する正規化項である．ただし，$\boldsymbol{h}_j^{(0)}$ は出力ではなく，API コールグラフの頂点 j に割り当てられた特徴量である．また，$\mathcal{N}_{in}(i) = \{(j,m) \mid e_{ji_m} \in E\}$，$\mathcal{N}_{out}(j) = \{(i,m) \mid e_{ji_m} \in E\}$，$c_{ji} = \sqrt{|\mathcal{N}_{out}(j)|} \cdot \sqrt{|\mathcal{N}_{in}(i)|}$ である．このことから，GCN は辺に割り当てられた特徴量を考慮せずにグラフ畳み込み演算を行うという制約を持つ．

2.3 関連研究

2.3.1 GCN を API コールグラフに適用する研究

GCN を API コールグラフに適用してマルウェアを判別する研究として Zhao ら [9]，Oliveira ら [10] の研究がある．

前者 [9] は，静的解析によりマルウェアのアセンブリコードから API を抽出し，API コールグラフを作成する．しかし，岩田らの調査によると，60%から 80%のマルウェアはパッカーで難読化されている [11]．また，マルウェアをアンパックするだけでなく，コードの挿入や置換，並び替えといった難読化を解読する必要がある [12]．特に，Moser らが提案した不透明な定数を生成する難読化手法は，静的解析を NP 困難であることが示されている [13] そのため，静的解析から API コールグラフの作成までの時間が長くなってしまい，大量のマルウェアが開発されている昨今の情勢に対処できない．

一方，後者 [10] は Cuckoo Sandbox を用いてマルウェアの動的解析を行い，そこから得られる解析レポートから API コール列を抽出することで API コールグラフを作成する．この手法により，GCN は LSTM と同等の判別精度

を発揮している．しかし，2.2 節で示したように GCN は頂点の特徴量のみで畳み込むという制約を持つため，API コールの引数を考慮できない．

2.3.2 API の引数を考慮するための研究

呼び出される API が同じであっても引数が異なれば，プログラムの動作は異なったものになる．例えば，ファイルをコピーする API の，コピー元，およびコピー先を指定する引数がそれぞれマルウェア，およびシステムディレクトリであれば，その API と引数はマルウェアの特徴を表現しているといえる．そのため，API の引数を考慮することで，マルウェアか否かを判別する特徴をより正確に学習できるようになる．API の引数の前処理に関する研究として Ahmed ら [14]，Rabadi ら [15] の研究が挙げられる．

Ahmed ら [14] は，平均や分散といった統計量で表現されたベクトルに変換している．この手法は，API コールの引数が変数やポインタという整数値であることに基づいている．しかし，この手法ではファイルの名前や URL を区別することができない．

Rabadi[15] らは API コール列を文書とみなして Bag of Words を適用することで特徴量を生成する．この特徴量を XGBoost へ入力することで，API コールの出現頻度に基づいて特徴を抽出する手法よりも高い F1-Score を達成している．しかし，Bag of Words は単語が存在するか否かのみを考慮する前処理であること，XGBoost は決定木の派生であることから，Rabadi らの手法はプログラムが持つ逐次，分岐，反復の 3 つの処理を明示的に考慮することができない．

3. 提案手法

3.1 提案の目的

本研究は，API コールグラフを用いたマルウェアの振る舞い検知において API コールの引数を考慮できるように拡張した EGCN を提案する．提案の目的は次の通りである．

- OS 上で実行されているプログラムは，いくつかの API を呼び出すことで，プログラムの機能を実現する．そのため，プログラムが呼び出す API からマルウェアか否かを判別する特徴を学習することができる．
- 動的解析のログから API コール列を抽出するため，難読化されたコードを解読してから API コール列を抽出する必要がある静的解析と比べてデータセットの収集が容易である．
- プログラムは逐次，分岐，反復の 3 つの処理を持つ．RNN 系は時系列処理であるため，逐次処理を考慮することができるが，分岐処理と反復処理を明示的に考慮することができない．一方，グラフデータは 3 つの処理を表現することができる．そのため，グラフデータを処理できる EGCN は，プログラムが持つ逐次処理に加えて分岐処理と反復処理も学習できる．
- EGCN は API の引数を考慮できるため，マルウェアか否かを決定する特徴をより正確に学習できるようになり，判別精度の向上が期待される．
- RNN は再帰構造を持つため，時間軸方向に展開すると多層なニューラルネットワークとみなせる，また，RNN の派生である LSTM や GRU はゲート構造も持つため，通常のニューラルネットワークと比べて計算が複雑である．これらにより，RNN は，ある時点の入力が出力にどのように影響するのかを考察することが難しい．一方，EGCN は畳み込みの対象となる頂点への有向辺と，その辺を持つ頂点のみが計算に関わる．そのため，RNN，およびその派生と比べて，頂点と辺の特徴量が出力にどのように影響するのかを考察しやすいと考えられる．
- RNN は前の時刻の出力を入力として受け取るため、各時刻の入出力を並列に処理することができない．一方，EGCN は各頂点の畳み込み演算を独立に行えるため，並列処理が可能である．これにより，学習の高速化が期待される．

3.2 提案手法の流れ

提案手法の流れを**図 1** に示す．提案手法は，(1)〜(4) 入力となるデータの作成，(5)〜(8) データの前処理，(9)〜(10) モデルの学習・推論の 3 つの段階から構成されている．

3.2.1 入力となるデータの作成

(1) Cuckoo Sandbox が出力する解析レポート (report.json)，あるいは他のデータセットから API コール列を抽出する．API の引数を含めるか否かで抽出の方法が異なるため，3.3 節で詳細を説明する．

(2) 引数なしの API コール列から，名前が重複する API を除いたときの API の総数を次数とする単位行列 H_{API} を作成する．この H_{API} の各行は，対応する頂点，すなわち API の名前の特徴量を表す．

(3) 引数ありの API コール列のデータと，引数なしの API コール列のデータを結合させる．

(4) k 分割交差検証を行うため，結合したデータを $k-1:1$ の比率で分割する．

3.2.2 データの前処理

(5) 3.4 節に従い，訓練データから API の引数のコーパスを作成する．ただし，分割した数だけコーパスを作成することに注意する．

(6) 3.4 節に従い，API コール列から特徴行列 H_{arg} を作成する．この H_{arg} の各行は，対応する辺，すなわち API コールの引数の特徴量を表す．

(7) API コール列から API コールグラフを作成する．

(8) 作成した特徴量 H_{API}, H_{arg} を API コールグラフの頂点，および辺に割り当てる．

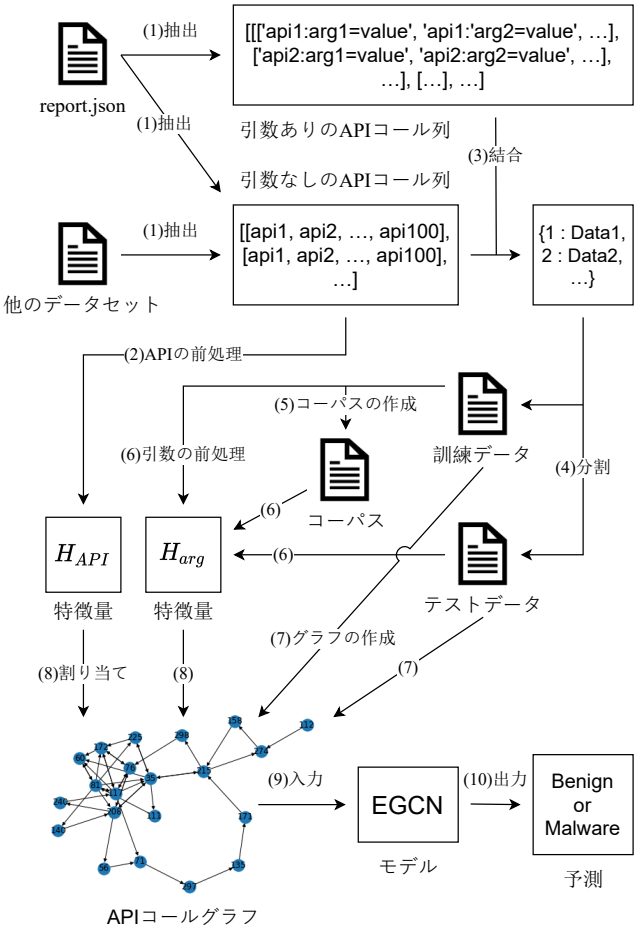

図 1 提案手法の概要

Fig. 1 The overview of the proposal.

3.2.3 モデルの学習・推論

(9) API コールグラフを GCN の入力として与える．

(10) GCN の出力に従ってパラメータの更新，あるいはマルウェアか否かの判別を行う．

3.3 API コール列の抽出

3.2.1 小節 (1) において，引数なしの API コール列を抽出する際は [10] と同じ条件とする．その条件を以下に示す．

(a) ループを避けるために，API コール列の最初の非連続の 100 個のみを用いる．

(b) 親プロセスから収集された API コール列のみを用いる．

一方，引数ありの API コール列を抽出する際は，以下に示すように条件 (a) を緩和した条件 (a') に従う．

(a') 同一の API コールが連続する部分列から，引数が異なる API コールのみを抽出した部分列を作成する．ただし，この部分列は 1 個の API コールとして数える．

これにより，引数ありの API コール列の長さが 100 を超える場合があるが，[10] と同じ条件で抽出し直すことで引数なしの API コール列と同一のものを作成できる．

3.4 API コールの引数の前処理

3.2.2 小節 (5), (6) における API コールの引数の前処理は，Rabadi らの手法 [15] に基づいている．その具体的な流れを以下に示す．

(i) Ex, A, W, ExA, ExW といった API の名前の接尾辞を削除する．

(ii) API コールに引数があれば，以下の前処理も行う．

(a) 整数 x に対して，$x > 0$ であれば 'pos' + $\log_2 x$，$x = 0$ であれば 'num0'，$x < 0$ であれば 'neg' + $\log_2(-x)$ とする．例えば，$x = 4$ であれば 'pos2' となる．

(b) ディレクトリのパスに対して，'System32' の文字列を含むのであれば 'sys_dir'，そうでなければ 'other_dir' とする．

(c) 拡張子に対して，'exe', 'dll', 'doc' といったマルウェアとして使用されやすい拡張子であれば対応する拡張子，そうでなければ 'other_file_type' とする．例えば，'sample.exe' であれば 'exe' となる．

(d) レジストリキーに対して，'BootExecute', 'Winlogon', 'Run', 'RunOnce' といった文字列を含めば対応する文字列，そうでなければ 'other_reg_key' とする．

(iii) 引数ごとに API の名前，引数の名前と値を結合し，単語 w = 'API_name:arg_name=arg_value' を作成する．

(iv) 作成した全ての単語を集合としてコーパスを作成する (図 1(5))．

(v) このコーパスを用いて訓練データやテストデータに対して Bag of Words を適用する．ただし，Rabadi らの手法 [15] とは異なり，1 つの API コールとその引数群 $d = (w_1, w_2 \cdots, w_n)$（$n$ は API コール d の引数の個数）を 1 つの文書とする．また．1 つの API コール列を 1 つの文書群 $D = (d_1, d_2 \cdots, d_T)$（$T$ は API コール列 D の長さ）とみなす．この文書群 D に対して Bag of Words を適用し，特徴行列 H_{arg} を作成する (図 1(6))．このとき，引数が無い，あるいは欠損している API コールに対応する行は $\mathbf{0}$ となるため，引数なしの API コール列のデータセットも活用することができる．

3.5 畳み込み演算の拡張

3.2.1 小節 (8) より，API コールグラフの頂点 j，および辺 e_{ji_k} にはそれぞれ特徴量 h_j, h_{ji_k} が割り当てられる．EGCN は，畳み込み演算を行う前に，辺 e_{ji_m} の特徴量 $h^{(0)}_{ji_m}$ を $\hat{h}^{(0)}_{ji_m} = \left(h^{(0)}_j \middle| h^{(0)}_{ji_m}\right)$ に更新する．ただし，$|$ はベクトルを結合する操作である．その後，式 1 の $h^{(0)}_j$ を $\hat{h}^{(0)}_{ji_m}$ に置き換えることで，辺の特徴量，すなわち API の引数を考慮することができるようになる．

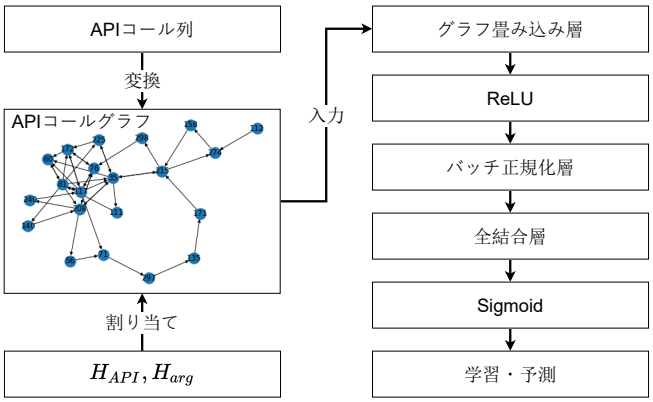

図 2 GCN の構成
Fig. 2 The structure of the Graph Convolutional Network.

3.6 GCN の構成

3.2 節の (1) から (10) を経て作成された API コールグラフを GCN に入力してマルウェアか否かの判別を行う．今回の実験で使用する GCN の構成を**図 2** に示す．この GCN は API コールグラフを入力とし，1 層のグラフ畳み込み層，ReLU，バッチ正規化層，全結合層，Sigmoid を経て出力する．この出力が 0 以上であればマルウェア，そうでなければ良性ソフトウェアと予測する．

4. 評価実験

本章では，使用するデータセットや実験の詳細，評価指標，実験環境について説明する．

4.1 データセット

本実験では，Oliveira が公開したデータセット [16]，および FFRI Dataset を使用する．どちらも Cuckoo Sandbox による動的解析のログのデータセットであるが，以下に示す違いがある．前者は，Cuckoo Sandbox が出力する解析レポートに対して 3.3 節で示した条件に従って引数なしの API コール列が抽出されている．また，データセットの内訳は，マルウェアが 42797 個，良性のソフトウェアが 1079 個である．一方，後者は，Cuckoo Sandbox が出力する解析レポートのデータセットとなっている．そのため，3.3 節で示した条件を満たすように解析レポートから API コール列を抽出する．その結果，引数あり，および引数なしの API コール列を 12446 個ずつ抽出することができた．また，その内訳を**表 1** に示す．両者のデータセットを結合させ，55243 個のマルウェア，および 1079 個の良性ソフトウェアの合計 56322 個の API コール列のデータセットを作成した．また，3.2.1 節 (2) の前処理を行った結果，API は 260 種類あることが確認された．この作成したデータセットに対して 10 分割交差検証を行い，性能を評価する．

4.2 使用するモデル

提案手法の性能を検証するため，EGCN, GCN, RNN, GRU, LSTM を実装してマルウェアか否かを判別させる実験を行う．それぞれのモデルのハイパーパラメータを**表 2** に示す．いずれのモデルも Adam を最適化アルゴリズムとして用いる．モデルの実装には Python 3.8.5, PyTorch 1.7.0 [17], CUDA 10.1 を用いる．また，EGCN，および GCN の実装には Deep Graph Library 7.0 [18] も用いる．

表 1 前処理の前後の FFRI Dataset の内訳
Table 1 The breakdown of FFRI dataset.

年 (解析環境)	前処理前	前処理後
2013	2640	2253
2014	3000	2388
2015	3000	1592
2016 (Windows 8.1)	8243	6213
2016 (Windows 10)	8243	0
2017	6251	0
合計	31377	12446

表 2 モデルのハイパーパラメータ
Table 2 The hyperparameter of the models.

	EGCN	GCN	LSTM	GRU	RNN
隠れ層の数	1	1	1	1	1
隠れ層の次元	8	32	64	64	32
学習係数	10^{-4}	10^{-4}	10^{-3}	10^{-3}	10^{-3}
正則化定数	10^{-3}	10^{-3}	0	10^{-4}	10^{-3}
ドロップアウト	0	0	0.5	0.5	0.5
バッチサイズ	64	64	128	128	128
バッチ正規化	あり	あり	なし	なし	なし

4.3 評価指標

本実験では，F1-Score，1Epoch 当たりの学習時間，F1-Score が一定値に収束するまでの学習回数，合計学習時間を評価指標として用いる．各評価指標の定義を以下に示す．

$$F1\text{-}Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$\text{1Epoch 当たりの学習時間} = \frac{\text{合計学習時間}}{\text{学習回数}}$$

ここで，1 回学習するごとに F1-Score，および学習時間を評価する．また，分散による影響を吸収するため，F1-Score に対して 5 回分の単純移動平均を求め，その中での最大値を判別精度として評価する．また，F1-Score が最大となったときを学習が収束したとみなす．これにより，合計学習時間を計算することができる．TP (True Positive)，FP (False Positive)，TN (True Negative) および FN (False Negative) の関係は**表 3** に示す通りである．

表 3　分類の結果
Table 3　The confusion matrix for binary classification.

		正解ラベル	
		Positive	Negative
予測	Positive	TP	FP
	Negative	FN	TN

表 4　実験環境
Table 4　The experimental setup.

CPU	Core i7-9750H 6core 2.60GHz
GPU	NVIDIA GeForce GTX 1650
Memory	DDR4-2666 SDRAM 16GB
OS	Windows 10

表 5　実験結果
Table 5　The result.

モデル	F1-Score	合計学習時間 [s]	学習回数	学習時間 [s]
EGCN	0.99662	7060	47	150.2
GCN	0.99733	2768	90	30.7
LSTM	0.99740	1215	376	3.2
GRU	0.99727	777	256	3.0
RNN	0.99510	1247	486	2.6

4.4　実験環境
プログラムが動作する環境は**表 4** に示す通りである．

4.5　実験結果
実験の結果を**表 5**, **図 3**, **図 4** に示す．EGCN は，GCN や LSTM，GRU と比べて F1-Score が低く，合計学習時間も長いという結果になった．また，図 4 より，EGCN の F1-Score の収束の仕方は GCN 似ているが，GCN と比べて低い F1-Score で収束が始まっている．そのため，時間を掛けて EGCN を学習させても，GCN を超える F1-Score が得られる可能性は低い．また，EGCN は 1Epoch 当たりの学習時間が最も長く，GCN の約 5 倍であった．GCN の判別精度は，GRU よりも僅かに高く，LSTM よりも僅かに低いという結果であった．しかし，合計学習時間は，LSTM の約 2.8 倍，GRU の約 3.6 倍という結果であった．したがって，EGCN，および GCN は，LSTM や GRU よりも優れた性能を発揮するとはいえない．

5.　考察
本章では，EGCN の判別精度，および学習速度が従来手法よりも下回った原因について考察する．

5.1　判別精度の考察
EGCN の判別精度が GCN よりも低かった原因として，パラメータの数が増えたことによる過学習が考えられる．

実験で使用された訓練データから抽出された API の個数，API と引数を結合した単語 w の個数はそれぞれ 260 個，平均 3928 個である．これにより，GCN の $W^{(0)}$ のパラメータの数は 8320 $(= 260 \times 32)$ 個であったが，EGCN では 33504 $(= (260 + 3928) \times 8)$ 個と増加している．一方で，パラメータの数が増えたことで，マルウェアの判別において冗長なパラメータも増えた可能性が考えられる．そのため，本実験で用いた EGCN の学習済みパラメータ $W^{(0)}$ を**図 5**, **図 6** に示すヒートマップで可視化した． 図 5, 図 6 より，所々に白線，すなわち **0** に近似できる行を確認できる．以降では，この白線が表す意味について考察する．

式 1 において，$(j', k') \in \mathcal{N}_{in}(i)$ かつ $W^{(0)}$ の $j' \leq 260$ 行目が **0** であるとき，$\boldsymbol{h}_{j'}^{(0)} W^{(0)} = \mathbf{0}$ である．これは，頂点 j' の特徴量，すなわち頂点 j に対応する API を考慮せずに頂点 i におけるグラフ畳み込み演算が行われることを意味している．同様に，$j' > 260$ 行目では，その行に対応する辺の特徴量，すなわち対応する API の引数を考慮せずにグラフ畳み込み演算が行われる．したがって，$W^{(0)}$ の j' 行目が **0** に近似できるとき，その行に対応する API，あるいは引数はマルウェアの判別において重要度が低いと解釈することができる．

EGCN の $W^{(0)}$ において **0** に近似できる行がどれほど存在するかを確認するために，$W^{(0)}$ の行ごとに L1 ノルムを計算した．API のみの分布，および引数のみの分布をそれ

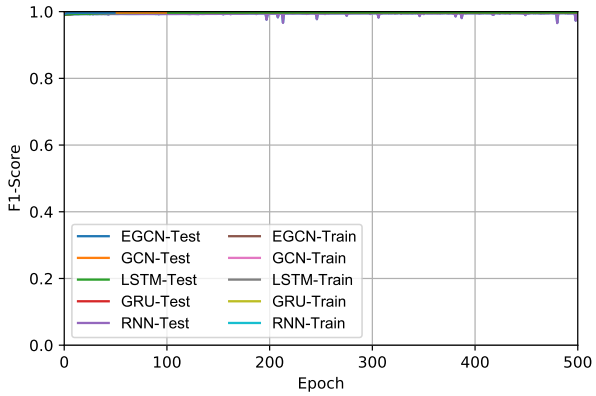

図 3　各回ごとの F1-Score
Fig. 3　The F1-Score at each epoch.

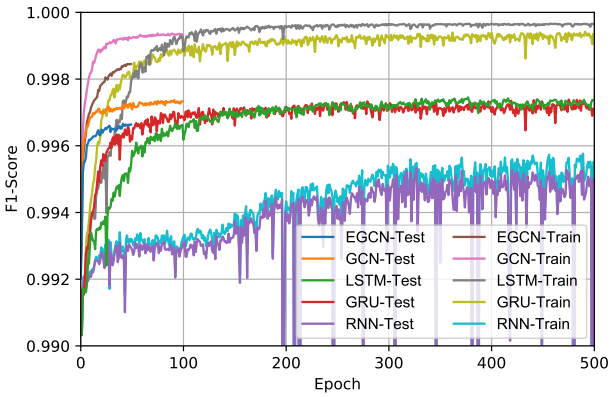

図 4　F1-Score（拡大）
Fig. 4　The F1-Score at each epoch (enlarged).

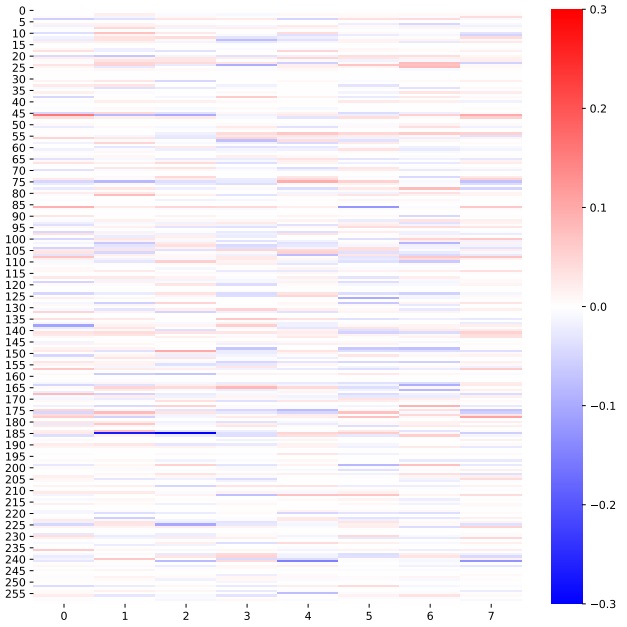

図 5 EGCN の $W^{(0)}$ のヒートマップ（API のみ）
Fig. 5 The heatmap of EGCN's $W^{(0)}$ (APIs only).

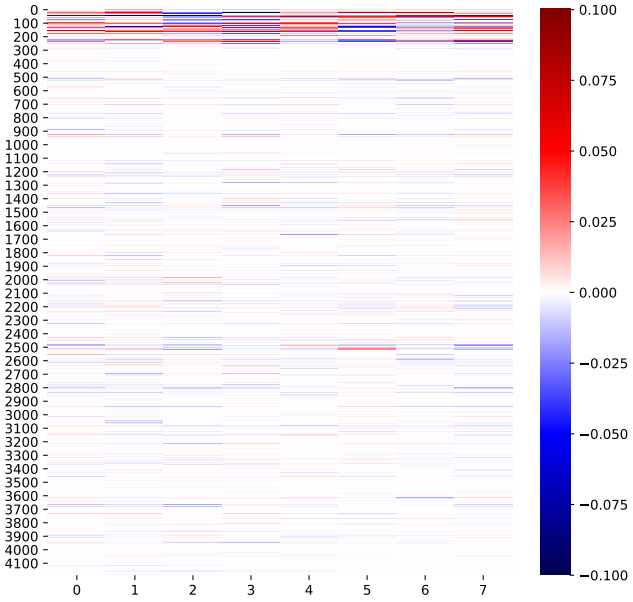

図 6 EGCN の $W^{(0)}$ のヒートマップ
Fig. 6 The heatmap of EGCN's $W^{(0)}$.

ぞれ**図 7**, **図 8** に示す．ここで，横軸は L1 のノルムの値，縦軸は L1 ノルムの値が 10^x 以上 10^{x+1} 未満の度数を表す．図 7, 図 8 より，L1 ノルムが 10^{-39} 未満と極端に小さい API，引数はそれぞれ 32 個，13 個あった．また，10^{-1} 以上の API，および引数はそれぞれ 133 個，13 個であり，API そのものと比べて API の引数に関するパラメータの絶対値が小さいといえる．このことから，マルウェアの判別において重要でない API，および引数が存在する．また，API の引数は，API そのものと比べるとマルウェアの判別において重要でないと考えられる．

本研究は，API に対しては One-Hot Encording，引数に

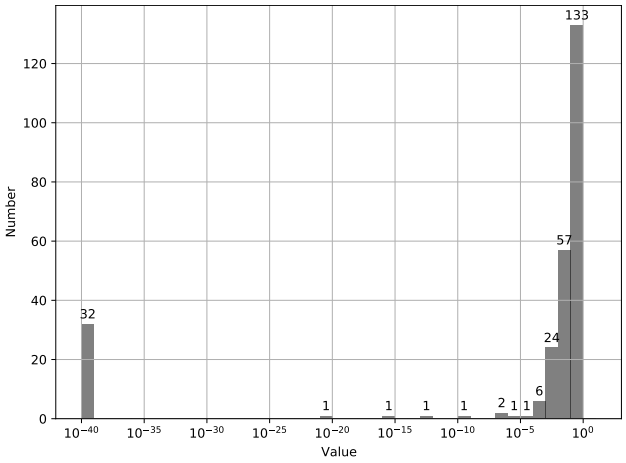

図 7 L1 ノルムの分布（API のみ）
Fig. 7 The distribution of L1 norm (APIs only).

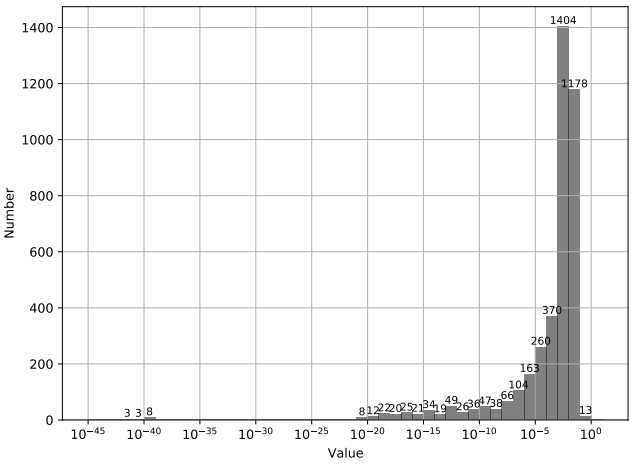

図 8 L1 ノルムの分布（引数のみ）
Fig. 8 The distribution of L1 norm (arguments only).

対しては Bag of Words を用いて特徴量を作成した．しかし，これらの前処理は全ての API，あるいは引数を平等に扱ってしまう．前段落で述べたように，API，およびその引数の重要度は異なるため，このことを考慮した前処理を行うことで判別精度が改善すると考えられる．

5.2 学習速度の考察

4.5 節の実験結果より，グラフ畳み込み層は LSTM 層や GRU 層と比べて処理に時間が掛かり，1Epoch 当たりの学習時間が長くなった．その原因として，疎行列積計算がグラフ畳み込み演算のボトルネックとなっている可能性が考えられる．疎行列積計算とは，疎行列と密行列との積を計算することである．ここで，疎行列の次元が数十から数百である場合，GPU の高い演算能力を十分に活かすことができない [19]．ここで，式 1 を行列で表現すると $H^{(1)} = B^{(0)} + A^T H^{(0)} W$ となる．上式より，頂点の数に対して辺の数が少ないグラフであるとき，隣接行列 A が疎行列となる，畳み込み演算が遅くなる．本研究で用いた API コールグラフの頂点の数は 260 個であるため，隣接行列は

260 次元である．また，3.4 節より，引数ありの API コール列から作成される API コールグラフの辺の数は高々 199 個である．したがって，API コールグラフの隣接行列のうち 99.7% 以上の要素が 0 の疎行列である．一方，$W^{(0)}$ は密行列であるため，本研究の EGCN，および GCN は疎行列積計算を行っており，この計算によって学習が遅くなったと考えられる．

5.3 データセットに関する制約

4.1 節より，本実験で使用したデータセットは，55243 個のマルウェア，および 1079 個の良性ソフトウェアの API コール列という不均衡なデータセットであった．特に，良性ソフトウェアに関しては，引数なしの API コール列のデータセットしか用いていない．また，データセットは 2019 年までに収集された動的解析のログであるため，モデルが最新のマルウェアの特徴を正しく学習しているとは限らない．

6. おわりに

本研究では，API コールグラフを用いたマルウェアの振る舞い検知において引数を考慮するように拡張した Extended GCN が，GCN や LSTM，GRU よりも性能を下回った原因を考察した．判別精度に関しては，EGCN の学習済みパラメータには零ベクトルが含まれていたことから，マルウェアの判別において重要でない API，および引数を余計に考慮していたと考えられる．また，学習速度に関しては，EGCN は数百次元の疎行列による疎行列積計算を行っていることが原因であると考えられる．

今後の課題としては，マルウェアの判別において重要でない API，および引数を除外する前処理の検討や，疎行列積計算を回避，あるいは高速化させる手法の検討，データセットの充実が挙げられる．

参考文献

[1] AV-Test: AV-ATLAS (online), available from ⟨https://portal.av-atlas.org/⟩ (accessed 2021-08-06).

[2] 三菱電機株式会社：不正アクセスによる個人情報と企業機密の流出可能性について（第 3 報）（オンライン），入手先 ⟨https://www.mitsubishielectric.co.jp/news/2020/0212-b.pdf⟩（参照 2021-08-06）．

[3] 日本電気株式会社：当社の社内サーバへの不正アクセスについて（オンライン），入手先 ⟨https://jpn.nec.com/press/202001/20200131_01.html⟩（参照 2021-08-06）．

[4] He, K., Zhang, X., Ren, S. and Sun, J.: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034 (online), DOI: 10.1109/ICCV.2015.123 (2015).

[5] GLUE: GLUE Benchmark (online), available from ⟨https://gluebenchmark.com/leaderboard⟩ (accessed 2021-08-10).

[6] SuperGLUE: SuperGLUE Benchmark (online), available from ⟨https://super.gluebenchmark.com/leaderboard⟩ (accessed 2021-08-10).

[7] SQuAD: The Stanford Question Answering Dataset (online), available from ⟨https://rajpurkar.github.io/SQuAD-explorer/⟩ (accessed 2021-08-10).

[8] Kipf, T. N. and Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks, *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, (online), available from ⟨https://openreview.net/forum?id=SJU4ayYgl⟩ (2017).

[9] lin Zhao, B., dong Liu, F., Shan, Z., hang Chen, Y. and Liu, J.: Graph Similarity Metric Using Graph Convolutional Network: Application to Malware Similarity Match, *IEICE Transactions on Information and Systems*, Vol. E102.D, No. 8, pp. 1581–1585 (online), DOI: 10.1587/transinf.2018EDL8259 (2019).

[10] Oliveira, A. and Sassi, R.: Behavioral Malware Detection Using Deep Graph Convolutional Neural Networks, (online), DOI: https://doi.org/10.36227/techrxiv.10043099.v1 (2019).

[11] 岩田吉弘，大坪雄平，萬谷暢崇：マルウェアに使用されるパッカーの経年変化に関する考察，コンピュータセキュリティシンポジウム 2019 論文集，Vol. 2019, pp. 934–939 (2019).

[12] You, I. and Yim, K.: Malware Obfuscation Techniques: A Brief Survey, *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, pp. 297–300 (online), DOI: 10.1109/BWCCA.2010.85 (2010).

[13] Moser, A., Kruegel, C. and Kirda, E.: Limits of Static Analysis for Malware Detection, *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pp. 421–430 (online), DOI: 10.1109/ACSAC.2007.21 (2007).

[14] Ahmed, F., Hameed, H., Shafiq, M. Z. and Farooq, M.: Using Spatio-Temporal Information in API Calls with Machine Learning Algorithms for Malware Detection, *Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence*, AISec '09, p. 55–62 (online), DOI: 10.1145/1654988.1655003 (2009).

[15] Rabadi, D. and Teo, S. G.: Advanced Windows Methods on Malware Detection and Classification, *Annual Computer Security Applications Conference*, ACSAC '20, p. 54–68 (online), DOI: 10.1145/3427228.3427242 (2020).

[16] Oliveira, A.: Malware Analysis Datasets: API Call Sequences, IEEE Dataport (online), available from ⟨https://dx.doi.org/10.21227/tqqm-aq14⟩ (accessed 2021-8-17).

[17] PyTorch (online), available from ⟨https://pytorch.org/⟩ (accessed 2021-08-16).

[18] Deep Graph Library (online), available from ⟨https://www.dgl.ai/⟩ (accessed 2021-08-16).

[19] 長坂侑亮，額田彰，小島諒介，松岡聡：GraphCNN 向けの疎行列積計算 Batch 最適化，研究報告ハイパフォーマンスコンピューティング（HPC），Vol.2018-HPC-167, No. 7, pp. 1–9 (2018).