Community Discovery of Attribution Trace Based on Deep Learning Approach

Jian Xu[1,2], Xiaochun Yun[1,2,3], Yongzheng Zhang[1,2](✉), and Zhenyu Cheng[1,2]

[1] Institute of Information Engineering,
Chinese Academy of Sciences, Beijing 100093, China
{xujian,zhangyongzheng,chengzhenyu}@iie.ac.cn
[2] School of Cyber Security, University of Chinese Academy of Sciences,
Beijing 100093, China
[3] National Computer Network Emergency Response Technical Team/Coordination
Center of China, Beijing 100093, China
yunxiaochun@cert.org.cn

Abstract. In order to prevent potential network crime and halt attackers' operation further, collecting information to profile attackers is helpful. Because this exposes the identity of attackers, as well as provides IOC (Indicator of Compromise) to confirm whether devices have been compromised. In this information searching procedure, finding unknown information based on the existing ones is of crucial importance, because it leads to a more comprehensive profile about the attackers. Usually, these information pieces about a particular attacker form a tight connected community. Thus, finding the correct community label for the new incoming information piece based on these existing ones is pivotal for iteratively discovering more unknown information about the attacker. To facilitate this process, we propose to adopt the promising deep learning method to community classification on attribution traces. First, we propose to employ deep learning on extracting attribution trace pattern and then use the fine-tuned DBN (Deep Belief Network) to model the existing communities. At last, we experimentally illustrate the effectiveness of the DBN model in finding the correct community labels by feeding it with test information pieces. The results demonstrate that deep learning is a powerful means for identifying the community label.

Keywords: Deep learning · Attribution trace · Network analysis
Community discovery

1 Introduction

With the highly developed global internet, a variety of network attacks are appearing daily, and this number is increasing [23,25]. To counter these threats, Network traceback is a sound method, because it directly leads to the exposure of attackers. For the real application, supposing to continuously monitor the APT1 organization [22], we collect information pieces about the attackers,

D. Naccache et al. (Eds.): ICICS 2018, LNCS 11149, pp. 342–357, 2018.
https://doi.org/10.1007/978-3-030-01950-1_20

such as malware MD5, IP address, email addresses used by them from public reports, Google, malware reverse engineering, and compromised device forensic. These information pieces are combined as traceback network to profile the APT1 organization and its attackers. We could exploit this traceback network to verify if machines are compromised by APT1 organization and its attackers through comparing the malware MD5 information pieces that are testified belonging to APT1 traceback network with the file MD5 found in end machine. If there exists a match, it indicates this machine has been compromised by this organization. The malware MD5 here we used are called the IOC (Indicator of Compromise). There are variety forms of IOCs, such as MD5, file existence, cookies, etc.

In general, tight connected information pieces are connected as relevant circles [24] and stay as a compact community. There exist more connections within the community rather than between communities. The more compact the community is, the more related these information pieces are to another. In the attribution trace network, these pieces within the community are strong relevant to the person or the organization we are investigating, while others may not much relevant. So as to continuously add new information pieces into this traceback network for comprehensive organization profiling, We iteratively research these information pieces within the community to discover other relevant but unknown pieces which possibly have been ignored hitherto by analyzers. We also have to determine whether the new incoming information piece belongs to this traceback network or not. Therefore, finding the correct community label for the information piece is of crucial importance as they may help to uncover previously unknown information about the attackers.

Deep learning, as the cutting-edge machine learning method, is very effective in extracting pattern. The pretraining selects better pattern layer by layer automatically. And these patterns extracted by deep learning are usually superior to hand-picked features [12]. Additionally, patterns extracted by pretraining is resistant to breakings launched by attackers. For example, in the past, once the features hand-picked by investigators were analyzed and understood, the attackers usually will modify their implemented features to escape the detection, which renders the previous research works ineffective and finally avoids the traceback mechanism. Based on these grounds, we employ pretraining to extract the community patterns from attribution trace network and model these communities through fine-tuning this model, powering the deep learning model to assign the new piece to the right community.

Our paper proposes a deep learning approach to find the correct community label for information pieces to fulfill the purpose of network attack traceback. To demonstrate the effectiveness of this approach, we use the Enron emails communication dataset to test the model trained by deep learning.

In summary, we make the following contributions to attack traceback in this paper:

- We utilize pretraining to extract connection patterns from both the connection weight and the connection structure of the trace network and demonstrate that the extraction of network trace pattern implemented by deep

learning, which learns optimized deep hierarchical representation. This overcomes the shortcomings accompanied with previous researches, which merely focus on individual features, such as vertex centrality, closeness centrality, eigenvector centrality and clustering coefficients.
- We validate the effectiveness of the deep learning model against the real email communication dataset. And hyper-parameters (i.e. $L_2 weight$ and sparsity), which greatly influence the model, are also discussed and optimized.

The rest of the paper is organized as follows. In Sect. 2, related literatures are reviewed, including the methods based on mesoscopic features, block-modeling and NMF (Nonnegative Matrix Factorization). Our deep learning approach are presented in Sect. 3. We outline our experiments in Sect. 4, and present experimental results. We close with a discussion about the work in Sect. 5.

2 Related Work

Community discovering has been extensively investigated in social network research area [1,3,5]. However, adopting community discovery to attribution trace network is a relatively emerging research area.

The traditional methods are usually based on the mesoscopic features, e.g. modularity, network vertex degree and connection weights. Seldom concerns have been given to applying deep learning for trace community discovery. Blondel et al. introduced an efficient modularity-based algorithm which finds communities in large networks [2].

Ferrara analyzed the community structure of the Facebook social network using the mesoscopic feature. It included the statistical features of the meta-network, such as density, average degree, connection weight, and it unveiled the communities representing the aggregation units where users stay together and interact [7]. Emilio Ferrara also presented LogAnalysis, which is a semi-supervised detection of criminal communities in networks reconstructed from phone call records. It analyzed degree centrality, vertex between centrality, closeness centrality, eigenvector centrality and clustering coefficient to profile the criminal communities. The system unveiled a few primary characteristics of criminal communities in real world phone call networks [8]. Most of their works focused on employing local features of vertex to detect community structures. Our work mainly utilizes the connection structure and connection weights of vertices.

De Meo et al. worked on methods to determine whether two vertices belong to the same community according to their similarity, which is based on the knowledge of common connected vertices or vertex group, as well as the analysis of social events in which users are involved [6]. They also proposed CONCLUDE (Complex Network Cluster Detection). It couples the accuracy of global approaches with the scalability of local methods by figuring out the edge importance which keeps the network connected. This edge centrality enables vertices mapping to Euclidean space. And the distance among the points in this space is employed to discover communities [15]. Their work is somewhat similar to ours, we both exploits the vertex similarity to determining whether two vertices belongs to the same community. However, they used features of the edge centrality, while we employ deep learning to extract representations from connection structure.

Chen et al. tried to uncover crime community by using hierarchical clustering, and they exploited block-modeling to confirm the inter-connection among communities [4]. Their method is mainly a community cluster, while ours devotes to train deep network to recognize vertices belonging to the same community.

He et al. [9] and Jin et al. [10] employed stochastic model to detect communities through nonnegative matrix factorization. It aimed to find a nonnegative membership matrix to reconstruct the adjacency matrix of the network. Their objective functions are usually based on square loss function and Kullback-Leibleer divergence. Their work used matrix factorization to discover community structures, while our approach can adapt the community model to new training vertices and continuously improve the community abstraction.

We are aware of Ishai Rosenberg' work, they proposed DeepAPT [20] to attribute malwares to its national developers using Deep Neural Networks (DNN). They recorded the running behaviors of malware in sandbox as raw input to the neural network, from which the DNN extracted features about the malwares. They tested set of 1,000 Chinese and Russian developed malwares, and achieved accuracy rate of 94.6%. Our work is different from theirs because we focus on profiling the attackers and their organization through continuously information mining. While their work concentrated on discovering the authorship for binary malware code.

3 Method

In this section, we propose the deep learning method, including the embedding preprocessing in Sect. 3.2, the deep learning in Sect. 3.3.

3.1 Definition

Attribution trace is defined as the abstract of attributable information piece. It is defined as a tuple, (v_i, v_j, e_{ij}), which contains two vertices v_i, v_j and their connection e_{ij}. By combining traces with the common vertex, we define the trace network as a network $G = (V, E)$, where V is the set of all vertices, and E is the set of all connections. The network is a collection of all vertices and their connections.

3.2 Vertex Embedding

The input dimension of the deep learning is constant in both training and testing stages. Thus, the vertices in the attribution trace network should map into a space where each vertex's feature dimension is the same. And this mapping function is requested to reserve the community characters of each vertex. To achieve this purpose, we utilize Deepwalk to embed vertices in the attribution trace network. The dimension of the embeddings is constant, which serves as the input of our deep learning model.

Deepwalk learns embedding of trace vertices by encoding their connection relations into a continuous vector space [18]. For each vertex in the trace network, the walk begins from itself, then adds a new vertex randomly chosen from the neighbors of the last visited vertex until the maximum walk length is met. Deepwalk then uses Word2vec to model and update the embedding of the vertices. Word2vec first represents each $v_i \in V$ as one-hot vector. Given a vertex walk $W^n = (v_0, v_1, \dots, v_n)$ in the random walks, where $v_i \in V$, and V is the set of trace vertices, Word2vec will maximize the $Pr(v_n | v_0, v_1, \dots, v_{n-1})$ over all the walks in the random walk set by exploiting neural network to predict the following vertex in each random walk [16]. After this process, the weights of hidden layer are used to present each vertex as a vector.

Deepwalk is a particularly computationally-efficient predictive model for learning vertex embeddings from the random walks, because if two vertices have the same connection context in the network, they tend to generate the similar random walks. It then embeds vertices in a continuous vector space where connection context similar vertices are mapped to nearby points. This captures the neighborhood similarity and community membership of each vertex in the attribution trace network.

The original implementation of Deepwalk does not exploit the connection weights among vertices. In our paper, we choose a new vertex in the random walk process with the probability proportional to their connection weights, which means if an adjacent vertex has greater weight than other adjacent vertices, it has a bigger chance that the chain walks to this vertex.

Algorithm 1. Deepwalk algorithm with connection weight

```
Input: $G(V, E)$, $dimension$, $walk.length$,
Output: $embeddings$
$walks \leftarrow \emptyset$
for all $v_i \in V$ do
    $walk \leftarrow \emptyset$
    $walk.add(v_i)$
    $v_{cur} \leftarrow v_i$
    for i in range(walk.length) do
        $v_{next} = rand(v_{cur}.adj, weights)$
        $walk.add(v_{next})$
        $v_{cur} \leftarrow v_{next}$
    end for
end for
$embeddings = word2vec(walks, dimension)$
```

The Deepwalk algorithm with connection weight is illustrated in Algorithm 1. It first implements random walks starting from each vertex in the network with the possibility proportional to the connection weight. And all the random walks generated in the first step constitute the walk set which the Word2vec model works on. Then Word2vec model consumes these random walks to update the vertex embeddings. In our research, we set the walk length to 15 vertices, and 200 walks are generated for each vertex. The final embedding dimension is 160.

3.3 Deep Learning

Deep learning is particularly helpful for extracting pattern from complex data [11]. This character especially suits the goal that we classify trace network community. Because the community character of attribution trace is a complex data format that is encoded as the connection structure and connection weight among different traces. The deep learning model extracts the pattern optimized for the community character through layer by layer pretraining. The training procedure of deep learning extracts the patterns which are perfect representations of vertex's community character. The training procedure contains two steps, pretraining and fine-tuning. The core unit of the pretraining stage is the autoencoder.

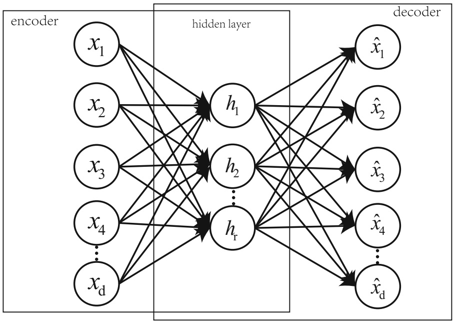


Fig. 1. Autoencoder unit

An autoencoder is comprised of an encoder and a decoder. The structure of autoencoder is depicted in Fig. 1. The encoder transfers the input to hidden pattern represented by the hidden layer output, while the decoder attempts to reverse this process by mapping the hidden pattern to its original input. We consider the vertex embeddings extracted by Deepwalk as $x \in \mathbb{R}^{D_x}$, then the autoencoder tries to replicate the input as $\hat{x}$ according to Eq. 1.

$$\hat{x} = h_d\{w_d(h_e(w_e x + b_e)) + b_d)\} \tag{1}$$

where h is the transfer function, w is the weight matrix and the b is the bias vector. The superscript e stands for the encoder layer, while the superscript d stands for the decoder layer.

The autoencoder learns its weights and biases by reconstructing the input through optimizing the cost function. Equation 2 is the cost function. The deep learning model updates the weights and biases through minimizing this cost function. The weights of the hidden layer learnt in this way are close to the global optimal weights, which are later used to transfer the input to the latent representations.

$$E = \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D}(x_{dn} - \hat{x}_{dn})^2 + \lambda \Omega_{weights} + \beta \Omega_{sparsity} \tag{2}$$

where N is the number of vertices in every training batch, D is the vertex dimension that is determined by Deepwalk's parameter, $\Omega_{weights}$ is the L_2 weight regularization and $\Omega_{sparsity}$ is the sparsity regularization. λ and β stand respectively for the coefficients of the L_2 weight regularization and the sparsity regularization.

The L_2 weight regularization is helpful in keeping the weights small. This is accomplished by summing the square of the weights. The overall term will be punished if this summation is large. This will prevent the model from overfitting the training traces [17].

$$\Omega_{weights} = \frac{1}{2}\sum_{l}^{L}\sum_{j}^{N}\sum_{i}^{D}(w_{ji}^{l})^2 \tag{3}$$

where D is the vertex dimension, N is the number of vertices in each training batch and L is the number of neurons in the hidden layer.

The sparsity regularization penalizes the activation of each hidden neuron, $\hat{\rho_i}$, deviating significantly from the hyper-parameter, ρ, by imposing penalty term to the cost function. It will punish the cost function when the average activation value, $\hat{\rho_i}$, of the reconstruct layer neuron is swayed much from the desired value, ρ.

$$\Omega_{sparsity} = \sum_{i=1}^{D}\rho log(\frac{\rho}{\hat{\rho_i}}) + (1-\rho)log(\frac{1-\rho}{1-\hat{\rho_i}}) \tag{4}$$

where ρ is a hyper-parameter to which we would like the average activation of each hidden neuron to be close, $\hat{\rho}$ is the activation of each hidden neuron. Here we use the Kullback-Leibler divergence function [13] as a difference measurement of $\hat{\rho_i}$ and ρ. The output of this function equals 0 when $\hat{\rho_i}$ and ρ are the same. D is the number of hidden neurons.

DBN is composed of a stack of encoders with a softmax layer as the final layer. These encoders are the front part of the pretrained autoencoders. The structure of DBN is demonstrated in Fig. 2. In DBN, each layer's input serves as the visible layer, output serves as the hidden layer. Each hidden layer serves as the visible layer of the next layer [21]. DBN fulfills the training process through three steps. The first step is pretraining, which goes layer by layer. It employs unsupervised training to update the weights of the hidden units in each layer,

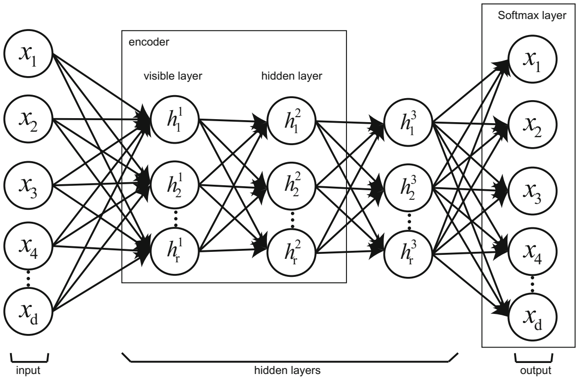


Fig. 2. DBN network

outputs the hidden patterns. During this step, each hidden layer in the DBN is considered as an autoencoder. Then it combines all the encoders trained in the first step and adds a softmax layer. The softmax layer squashes the output of the last hidden layer into vector $\sigma(z)$ of real values, where each entry is in the range (0, 1), and all the entries add up to one. Equation 5 is the softmax function.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad for\ j = 1, \ldots, K \tag{5}$$

After the pretraining stage, a fine-tuning process is launched using the training traces and their community labels in a supervised training manner to re-train the whole neural network. This process treats all layers of the stacked autoencoders as a single model and precisely adjust the model weights to fit the training traces. This fine-tuning process utilizes the cross-entropy as the cost function, which is displayed in Eq. 6.

$$E(W, b) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \ln \hat{y}_i + (1 - y_i)\ln(1 - \hat{y}_i)\right] \tag{6}$$

where N is the number of items in each training batch, the sum is over all training traces, and y is the corresponding target community class label. $\hat{y}$ is the predicted community label of the deep learning model.

In this work, we use the backpropagation algorithm to compute the gradients for all layers displayed through Eqs. 7 to 10.

$$\delta^{n_l} = -(\nabla_{a^{n_l}} E) \bullet f^{'}(z^{(n_l)}) \tag{7}$$

$$\delta^{l} = [(W^{l})^{T}\delta^{(l+1)}] \bullet f^{'}(z^{(l)}) \tag{8}$$

$$\nabla_{W^{(l)}} E(W, b; x, y) = \delta^{(l+1)}(a^{(l)})^{T} \tag{9}$$

$$\nabla_{b^{(l)}} E(W, b; x, y) = \delta^{(l+1)} \tag{10}$$

Where $\bullet$ denotes the element-wise product operator, $a^{(l)}$ is the output pattern vector in layer l, δ measures how much the node accounts for any errors in the final community output. $f(z)$ denotes the activation function of each layer. The algorithm first performs a feedforward pass. Outputs, $a^{(l)}$, of the hidden layer as well as the final are computed using each layer's forward propagation equation. Then $\nabla_W E(W,b;x,y)$ and $\nabla_b E(W,b;x,y)$ are the partial derivatives and they are calculated from the last layer back to the first layer. Finally, the W and b are updated according to Eqs. 11 and 12.

$$W_{ij} = W_{ij} - \alpha \frac{\partial}{\partial W_{ij}} E(W,b) \tag{11}$$

$$b_i = b_i - \alpha \frac{\partial}{\partial b_i} E(W,b) \tag{12}$$

After this training process, the network predict the labels for the test vertices.

4 Experiment

In this section, we first introduce the dataset we are going to investigate, and then we prune the dataset to extract the trace network. Deepwalk will be utilized to embed these vertices in the trace network. And the ground truth is obtained by applying modularity algorithm. We also discuss the optimization of the hyper-parameters of the deep learning model. And the model is trained from the test traces by applying these parameters. We finally present the deep learning's prediction result.

In this work, we use the Enron email dataset to prove the effectiveness of the proposed method [14]. This dataset contains 517,431 emails from 150 users distributed in 3,500 folders. Each email holds the sender and the receiver's email address, sent date and time, subject, email content and technical details about the email. The 150 users are mostly the Enron senior management members. We extract the email addresses and their connections from the raw email contents, and the extracted dataset contains 79,562 distinctive email addresses and 310,976 communications among them. After this, we further purge this dataset by dropping these emails that are not ended with @enron.com, limiting this investigation to the Enron company. By this time, the dataset is left with 32,190 addresses and 200,534 connections. Each of the connection bears a weight standing for the number of emails communicated by the two email addresses. And we consider the communication undirected, which means the connection weight is the summation number of two direction sent mails. The summary of the dataset is illustrated in Table 1.

We combine these traces in this pruned dataset to compose the trace network we are going to investigate. To evaluate the deep learning model's effectiveness, we employ the modularity algorithm to partition the traces in order to act as the ground truth of our deep learning approach. Modularity separates vertices to communities according to the density of connection within communities. It

Table 1. Dataset summary

Dataset	Vertices	Connections
Original dataset	79,562	310,976
Restrain to @enron.com	32,190	200,534

assumes that a community contains dense connections between vertices within the community, while there are sparse connections among different communities. The community modularity is defined in Eq. 13

$$Q = \frac{1}{2m} \sum_{vw} \left[A_{vw} - \frac{k_v k_w}{2m} \right] \delta(c_v, c_w) \tag{13}$$

In the equation, k_v is the degree of v and k_w is the degree of w. m is the total number of connections in the trace network; A_{vw} is the actual number of connections between vertex k_v and k_w. Q is the modularity index. $\delta(c_v, c_w)$ is a function that equals 0, when vertices v and w are in different communities, and equals 1, when they reside in the same community. c_v and c_w are the community label for v and w, respectively.

The modularity algorithm tries to assign traces to different communities by maximizing the modularity index according to Eq. 13. We employ this modularity algorithm to partition the Enron dataset into communities. The communities found by this algorithm is depicted in Fig. 3.

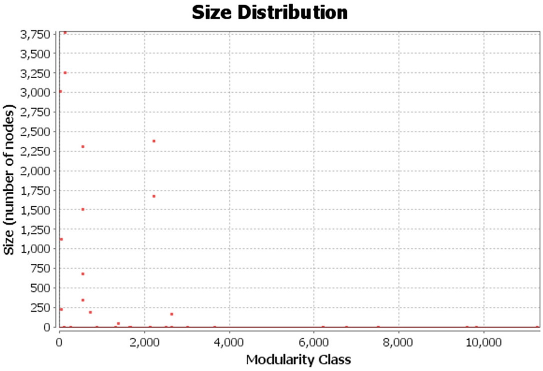


Fig. 3. Modularity based communities

From Fig. 3, we could figure out that most communities rarely contain email addresses. Most traces are included in a few large communities. More precisely, it is shown that the 10 biggest communities each contain over 250 vertices. Additionally, the 149 senior members (There exists a duplicated address in the original 150 email addresses) are scattered in these 10 communities. Based on this

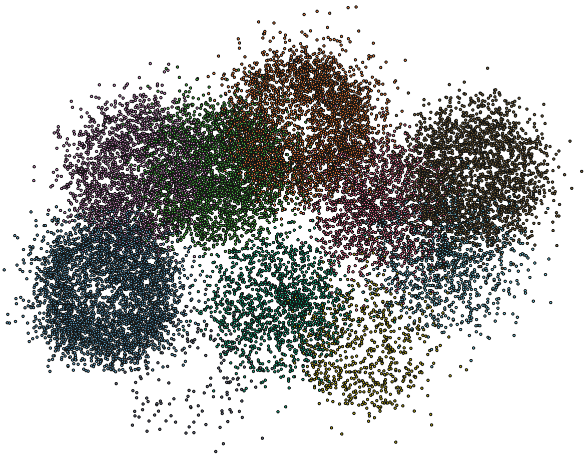


Fig. 4. Email communities

observation, we further limit the dataset to the 10 largest communities. These vertices pertaining to the 10 communities are reserved. It leaves us a dataset with 19,995 vertices and 196,197 connections. We apply this dataset as the ground truth to evaluate the effectiveness of community discovering based on the deep learning approach. Figure 4 is an illustration of the communities separated by modularity algorithm. Table 2 is a breakdown of the statistics about the 10 communities.

The DBN we employed is made up of two encoders as the hidden layers and a softmax layer as the output layer. The number of neurons for the first and second hidden layers are 100 and 80 respectively. The summary of the network parameters is displayed in Table 3.

Table 2. 10 large communities

Community ID	Email addresses	Senior manager emails
1	3,029	55
20	1,133	41
93	3,779	9
102	3,263	2
510	696	1
513	2,322	16
514	1,514	4
2191	1,689	7
2198	2,390	13
2618	180	1

Table 3. DBN parameter

Parameter		Value
Hidden layer	Train function	trainscg
	Cost function	msesparse
	L_2weight	1.5583e−5
	Sparsity	0.0485
Softmax layer	Train function	trainscg
	Cost function	crossentropy
Fine tune	Train function	trainscg
	Cost function	crossentropy

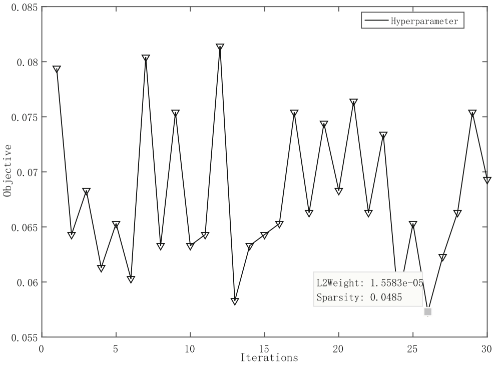


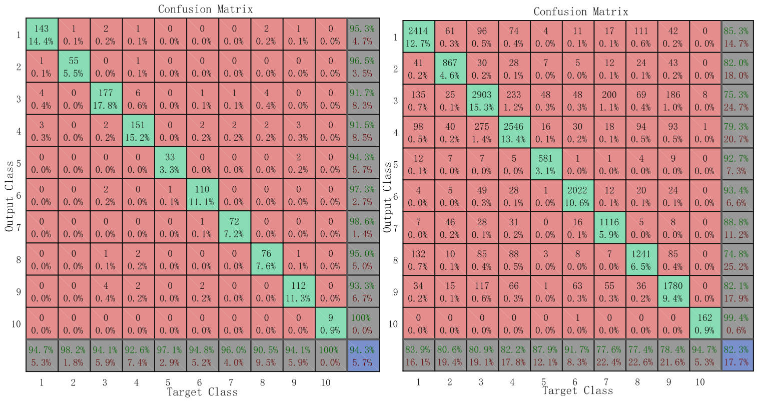
Fig. 5. Hyperparameters

We use scaled conjugate gradient backpropagation (SCG) as the training function for all layers. It updates weights along the conjugate directions which produces a faster convergence. The msesparse is a mean squared error function adjusted by L_2 weight and sparsity regularization according to Eq. 2 we have discussed in Sect. 3.3. The softmax layer and the final fine-tune stage are both supervised learning process, thus we adopt cross-entropy function to measure the difference between the prediction and the target.

The hyper-parameters, L_2weight Regularization and Sparsity Regularization, influence the prediction greatly, which is significant for the DBN to work properly. Thus, we conduct experiment to find the optimal values by trying different configurations. The search range for L_2weight Regularization is [1e−5, 1e−4] and Sparsity Regularization is [1e−5, 1e−1]. This optimization process iterates 30 times. The results are depicted in Table 4. Figure 5 is the plot of the objective value we are trying to minimize.

Table 4. Hyper-parameter optimization

Iter	L_2weight	Sparsity	Objective	Iter	L_2weight	Sparsity	Objective	Iter	L_2weight	Sparsity	Objective
1	9.43E−05	0.000104	0.0794	11	1.05E−05	0.00893	0.0643	21	1.02E−05	0.00679	0.0764
2	5.73E−05	0.0272	0.0643	12	1.71E−05	0.0317	0.0814	22	1.01E−05	0.00514	0.0663
3	1.26E−05	0.000397	0.0683	13	1.01E−05	0.00677	0.0583	23	1.01E−05	0.00835	0.0734
4	1.59E−05	0.0309	0.0613	14	1.54E−05	0.0273	0.0633	24	1.01E−05	0.00513	0.0583
5	4.61E−05	0.0394	0.0653	15	1.02E−05	0.00458	0.0643	25	4.59E−05	0.0393	0.0653
6	1.00E−05	0.00846	0.0603	16	1.56E−05	0.0486	0.0653	**26**	**1.56E−05**	**0.0485**	**0.0573**
7	1.01E−05	0.0173	0.0804	17	4.38E−05	0.0134	0.0754	27	4.31E−05	0.00709	0.0623
8	4.27E−05	0.00707	0.0633	18	1.00E−05	0.00622	0.0663	28	4.37E−05	0.00715	0.0663
9	3.36E−05	0.00964	0.0754	19	1.02E−05	0.00772	0.0744	29	1.61E−05	0.0309	0.0754
10	1.04E−05	0.00833	0.0633	20	1.02E−05	0.00657	0.0683	30	5.78E−05	0.0271	0.0693



(a) 95% training vertices

(b) 5% training vertices

Fig. 6. Recognition result

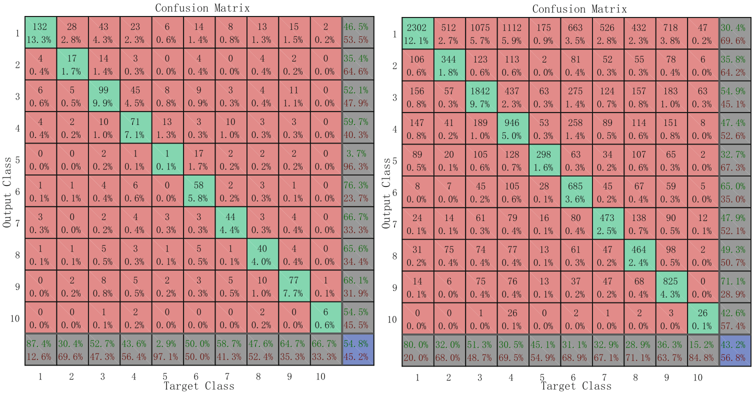
From Fig. 5 and Table 4, we could conclude the proper values of L_2weight and Sparsity are 1.5583e−05 and 0.0485, the corresponding minimized error objective is 0.0573.

To evaluate the model, we randomly choose 95% of 19,995 vertices from each of the 10 communities to train the DBN network, and then we use the remaining 5% vertices to test the prediction of the learnt deep learning model. The result is quiet promising, and it is demonstrated in Fig. 6(a). The model predicts the community label correctly for 94.3% of test vertices.

And then we further delve into evaluating the effectiveness of this deep learning network when it is fed with small percent of training vertices. We randomly choose only 5% of 19,995 vertices from each of the 10 communities as the training samples in this configuration. And the result is promising, the deep learning network performs very well even under this harsh condition. The result is demonstrated in Fig. 6(b). It assigns the community label correctly for 82.3% of 18,996 test vertices.

To demonstrate the effectiveness of deep learning approach, we compare our approach with the kNN (k nearest neighbor) classification [19]. We set the parameter of kNN to 4 neighbors. The same training and testing dataset setting is used to test kNN. The results are presented in Fig. 7(a) and (b). The overall accuracy is 54.8% when kNN is trained with 95% of vertices, and it drops to 43.2% when trained with 5% of vertices.

The accuracy of kNN depends on the parameter choice of k nearest neighbors. Usually, larger value reduces effect of the noise, but make boundaries between classes less distinct. To mitigate the effects of this parameter, we alternate the nearest neighbors from 1 to 10 in order to test the capability of kNN. The result is in Table 5. It shows the accuracy fluctuates around 55%, and the kNN can at best achieve accuracy of 55.68% when the nearest neighbors are set to 7.



Confusion Matrix

Output Class

	1	2	3	4	5	6	7	8	9	10	
1	132 13.3%	28 2.8%	43 4.3%	23 2.3%	6 0.6%	14 1.4%	8 0.8%	13 1.3%	15 1.5%	2 0.2%	46.5% 53.5%
2	4 0.4%	17 1.7%	14 1.4%	3 0.3%	0 0.0%	4 0.4%	0 0.0%	4 0.4%	2 0.2%	0 0.0%	35.4% 64.6%
3	6 0.6%	5 0.5%	99 9.9%	45 4.5%	8 0.8%	9 0.9%	3 0.3%	4 0.4%	11 1.1%	0 0.0%	52.1% 47.9%
4	4 0.4%	2 0.2%	10 1.0%	71 7.1%	13 1.3%	3 0.3%	10 1.0%	3 0.3%	3 0.3%	0 0.0%	59.7% 40.3%
5	0 0.0%	0 0.0%	2 0.2%	1 0.1%	1 0.1%	17 1.7%	2 0.2%	2 0.2%	2 0.2%	0 0.0%	3.7% 96.3%
6	1 0.1%	1 0.1%	4 0.4%	6 0.6%	0 0.0%	58 5.8%	2 0.2%	3 0.3%	1 0.1%	0 0.0%	76.3% 23.7%
7	3 0.3%	0 0.0%	2 0.2%	4 0.4%	3 0.3%	3 0.3%	44 4.4%	3 0.3%	4 0.4%	0 0.0%	66.7% 33.3%
8	1 0.1%	1 0.1%	5 0.5%	3 0.3%	1 0.1%	5 0.5%	1 0.1%	40 4.0%	4 0.4%	0 0.0%	65.6% 34.4%
9	0 0.0%	2 0.2%	8 0.8%	5 0.5%	2 0.2%	3 0.3%	5 0.5%	10 1.0%	77 7.7%	1 0.1%	68.1% 31.9%
10	0 0.0%	0 0.0%	1 0.1%	2 0.2%	0 0.0%	0 0.0%	0 0.0%	2 0.2%	0 0.0%	6 0.6%	54.5% 45.5%
	87.4% 12.6%	30.4% 69.6%	52.7% 47.3%	43.6% 56.4%	2.9% 97.1%	50.0% 50.0%	58.7% 41.3%	47.6% 52.4%	64.7% 35.3%	66.7% 33.3%	54.8% 45.2%

Target Class

(a) 95% training vertices, k=4

Confusion Matrix

Output Class

	1	2	3	4	5	6	7	8	9	10	
1	2302 12.1%	512 2.7%	1075 5.7%	1112 5.9%	175 0.9%	663 3.5%	526 2.8%	432 2.3%	718 3.8%	47 0.2%	30.4% 69.6%
2	106 0.6%	344 1.8%	123 0.6%	113 0.6%	2 0.0%	81 0.4%	52 0.3%	55 0.3%	78 0.4%	6 0.0%	35.8% 64.2%
3	156 0.8%	57 0.3%	1842 9.7%	437 2.3%	63 0.3%	275 1.4%	124 0.7%	157 0.8%	183 1.0%	63 0.3%	54.9% 45.1%
4	147 0.8%	41 0.2%	189 1.0%	946 5.0%	53 0.3%	258 1.4%	89 0.5%	114 0.6%	151 0.8%	8 0.0%	47.4% 52.6%
5	89 0.5%	20 0.1%	105 0.6%	128 0.7%	298 1.6%	63 0.3%	34 0.2%	107 0.6%	65 0.3%	2 0.0%	32.7% 67.3%
6	8 0.0%	7 0.0%	45 0.2%	105 0.6%	28 0.1%	685 3.6%	45 0.2%	67 0.4%	59 0.3%	5 0.0%	65.0% 35.0%
7	24 0.1%	14 0.1%	61 0.3%	79 0.4%	16 0.1%	80 0.4%	473 2.5%	138 0.7%	90 0.5%	12 0.1%	47.9% 52.1%
8	31 0.2%	75 0.4%	74 0.4%	77 0.4%	13 0.1%	61 0.3%	47 0.2%	464 2.4%	98 0.5%	2 0.0%	49.3% 50.7%
9	14 0.1%	6 0.0%	75 0.4%	76 0.4%	13 0.1%	37 0.2%	47 0.2%	68 0.4%	825 4.3%	0 0.0%	71.1% 28.9%
10	0 0.0%	0 0.0%	1 0.0%	26 0.1%	0 0.0%	2 0.0%	1 0.0%	2 0.0%	3 0.0%	26 0.1%	42.6% 57.4%
	80.0% 20.0%	32.0% 68.0%	51.3% 48.7%	30.5% 69.5%	45.1% 54.9%	31.1% 68.9%	32.9% 67.1%	28.9% 71.1%	36.3% 63.7%	15.2% 84.8%	43.2% 56.8%

Target Class

(b) 5% training vertices, k=4

Fig. 7. kNN recognition result

Table 5. kNN optimization (95% training vertices)

Neighbors	1	2	3	4	5
Accuracy	54.47%	51.46%	55.08%	54.77%	55.58%
Neighbors	6	**7**	8	9	10
Accuracy	55.58%	**55.68%**	54.87%	54.57%	54.47%

Seen from results of both deep learning and kNN, deep learning is stronger in extracting community patterns and finding the right community for new traces with great higher accuracy.

5 Conclusion

In conclusion, we proposed a deep learning based approach to discover the community label of attribution traces. The experimental results show that although trained with a small training set, this approach still could produce the promising result. It demonstrates that the deep learning method is applicable in the network-based trace analysis, and it would become a prominent method for trace analysis because of the following reasons. First, deep learning methods use pretraining to automatically extract the patterns, which are hand-picked by experts in the past. The extracted patterns usually perform better than the hand-picked patterns, because the pretraining considers all features that could be utilized to generate patterns. Second, this pretraining and classifying process are not easy to break. Thus, it is hard for attackers to modify features to avoid being detected, so attacker will not come out with countermeasures to avoid this traceback mechanism. It is worth noting that we used relatively small deep learning structure with 2 hidden layers of 100 and 80 respectively. The result would become much more promising if we deeper the layers and enlarge the neurons with more training iterations. In our future works, we would further rank vertex importance within the community. Rather than limited to the email communication connections, we also would consider to construct the trace network from variety of connections, such as sharing the same IP address, co-authoring the malware, attending the conference, working for the same organization, etc.

Acknowledgment. This work was supported by the National Natural Science Foundation of China (No. U1736218).

References

1. Bedi, P., Sharma, C.: Community Detection in Social Networks. Wiley, Hoboken (2016)
2. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech. **2008**(10), 155–168 (2008)
3. Catanese, S., Ferrara, E., Fiumara, G.: Forensic analysis of phone call networks. Soc. Netw. Anal. Min. **3**(1), 15–33 (2013)
4. Chen, H., Chung, W., Xu, J.J., Wang, G., Qin, Y., Chau, M.: Crime data mining: a general framework and some examples. Computer **37**(4), 50–56 (2004)
5. Chopade, P., Zhan, J., Bikdash, M.: Node attributes and edge structure for large-scale big data network analytics and community detection. In: IEEE International Symposium on Technologies for Homeland Security, pp. 1–8 (2015)
6. De Meo, P., Ferrara, E., Fiumara, G.: Finding similar users in Facebook, pp. 303–324 (2011)
7. Ferrara, E.: A large-scale community structure analysis in Facebook. EPJ Data Sci. **1**(1), 1–30 (2012)
8. Ferrara, E., Meo, P.D., Catanese, S., Fiumara, G.: Detecting criminal organizations in mobile phone networks. Expert Syst. Appl. **41**(13), 5733–5750 (2014)
9. He, D., Liu, D., Jin, D., Zhang, W.: A stochastic model for detecting heterogeneous link communities in complex networks (2015)
10. Jin, D., Chen, Z., He, D., Zhang, W.: Modeling with node degree preservation can accurately find communities. New Media Soc. **18**(7), 1293–1309 (2016)
11. Krizhevsky, A., Hinton, G.E.: Using very deep autoencoders for content-based image retrieval. In: Proceedings of the European Symposium on Artificial Neural Networks, ESANN 2011, Bruges, Belgium, 27–29 April 2011 (2012)
12. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: International Conference on Neural Information Processing Systems, pp. 1097–1105 (2012)
13. Kullback, S., Leibler, R.A.: On information and sufficiency. Ann. Math. Stat. **22**(1), 79–86 (1951)
14. Leskovec, J., Lang, K.J., Dasgupta, A., Mahoney, M.W.: Community structure in large networks: natural cluster sizes and the absence of large well-defined clusters. Internet Math. **6**(1), 29–123 (2008)
15. Meo, P.D., Ferrara, E., Fiumara, G., Provetti, A.: Mixing local and global information for community detection in large networks. J. Comput. Syst. Sci. **80**(1), 72–87 (2014)
16. Mikolov, T., et al.: Efficient estimation of word representations in vector space. In: International Conference on Learning Representations, pp. 1–12 (2013)

17. Olshausen, B.A., Field, D.J.: Sparse coding with an overcomplete basis set: a strategy employed by V1? Vis. Res. **37**(23), 3311–3325 (1997)
18. Perozzi, B., Al-Rfou, R., Skiena, S.: DeepWalk: online learning of social representations, pp. 701–710 (2014)
19. Peterson, L.: K-nearest neighbor. Scholarpedia **4**(2), 1883 (2009)
20. Rosenberg, I., Sicard, G., David, E.O.: DeepAPT: nation-state APT attribution using end-to-end deep neural networks. In: Lintas, A., Rovetta, S., Verschure, P.F.M.J., Villa, A.E.P. (eds.) ICANN 2017. LNCS, vol. 10614, pp. 91–99. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68612-7_11
21. Schölkopf, B., Platt, J., Hofmann, T.: Greedy layer-wise training of deep networks. In: International Conference on Neural Information Processing Systems, pp. 153–160 (2006)
22. Segal, A.: Mandiant: APT1: exposing one of china's cyber espionage units
23. Wheeler, D.A., Larsen, G.N.: Techniques for cyber attack attribution (2003)
24. Xu, J., Yun, X., Zhang, Y., Sang, Y., Cheng, Z.: NetworkTrace: probabilistic relevant pattern recognition approach to attribution trace analysis. In: 2017 IEEE Trustcom/BigDataSE/ICESS, pp. 691–698, August 2017. https://doi.org/10.1109/Trustcom/BigDataSE/ICESS.2017.301
25. Yao, S., Chen, J., Du, R., Deng, L., Wang, C.: A survey of security network coding toward various attacks. In: IEEE International Conference on Trust, Security and Privacy in Computing and Communications, pp. 252–259 (2014)