applied sciences

MDPI

Article

A Graph Representation Learning Algorithm for Low-Order Proximity Feature Extraction to Enhance Unsupervised IDS Preprocessing

Yiran Hao [1,2], Yiqiang Sheng [1,2,*] and Jinlin Wang [1,2]

[1] National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China; haoyr@dsp.ac.cn (Y.H.); wangjl@dsp.ac.cn (J.W.)

[2] School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing 100049, China

* Correspondence: shengyq@dsp.ac.cn; Tel.: +86-1-312-116-8320

Received: 12 September 2019; Accepted: 18 October 2019; Published: 22 October 2019

Featured Application: We use the proposed packet2vec learning algorithm for IDS preprocessing, the basic steps of IDS are as follows. First, the originally collected traffic is split into packets to be truncated into fixed length. Next, the packet2vec learning algorithm is used to obtain local proximity structure features of the packet for preprocessing. Then, the original features of the packet are combined with the local proximity features as the input of deep auto-encoder for IDS. Finally, the accuracy was evaluated with the detection rate in IDS. In addition, the model proposed in this paper can be deployed to the enterprise gateway, dynamically monitor network activities, and connect with the firewall to protect the enterprise's network from attacks. It can be deployed in a cloud computing environment or a software-defined network to classify traffic, and monitor network behavior and alerts in real time. It can be deployed into a network security situational awareness system for prediction and visualization through spatial feature extraction.

Abstract: Most existing studies on an unsupervised intrusion detection system (IDS) preprocessing ignore the relationship among packets. According to the homophily hypothesis, the local proximity structure in the similarity relational graph has similar embedding after preprocessing. To improve the performance of IDS by building a relationship among packets, we propose a packet2vec learning algorithm that extracts accurate local proximity features based on graph representation by adding penalty to node2vec. In this algorithm, we construct a relational graph G' by using each packet as a node, calculate the cosine similarity between packets as edges, and then explore the low-order proximity of each packet via the penalty-based random walk in G'. We use the above algorithm as a preprocessing method to enhance the accuracy of unsupervised IDS by retaining the local proximity features of packets maximally. The original features of the packet are combined with the local proximity features as the input of a deep auto-encoder for IDS. Experiments based on ISCX2012 show that the proposal outperforms the state-of-the-art algorithms by 11.6% with respect to the accuracy of unsupervised IDS. It is the first time to introduce graph representation learning for packet-embedded preprocessing in the field of IDS.

Keywords: graph representations; instruction detection; local proximity features; packet2vec; packet preprocessing; penalty

1. Introduction

Intrusion detection systems (IDSs) [1–3] have been introduced to monitor the network status, assess the security status, and take appropriate precautionary measure before the attack with serious

consequences. IDSs are classified into signature-based detection [4] and anomaly-based detection [5]. Signature-based intrusion detection has the ability to detect known attack behavior. Anomaly-based detection has the ability to detect known attack behavior and unknown attack behavior by analyzing the features of network packets. Anomaly-based detection requires finding features that accurately characterize network packets. However, the network packet preprocessing methods suffer from low performance to extract features [6].

One of the main reasons to lead the above situation is that the features extracted by the existing unsupervised algorithms [7–9] are not accurate to preprocess network packets. The network packets in the existing unsupervised algorithms [7–9] are considered to be independent. These algorithms ignore the similarity relationship between network packets. According to homophily hypothesis [10], it can be seen that network packets with strong similarity belong to the same class and are closer in the similarity relational graph. In addition, network packets with strong similarity have similar local proximity embedding [11]. Skip-gram [12] can be used to embed the local proximity features of the similarity relationships between network packets. Therefore, it can be concluded that the analysis of the similarity relationship between network packets has the ability to be more flexible and accurately determine whether the network packet has an intrusion. Moreover, effective preprocessing is the basis for improving the performance of the entire IDS because the network packet features obtained by preprocessing directly affect the final performance of the IDS. Graph representation approaches have a good potential for achieving effective similarity relationship representation of network packet. Therefore, this paper proposes a packet2vec learning algorithm preprocessing network packet, which is an unsupervised preprocessing algorithm based on graph representation learning. The algorithm uses packet2vec for preprocessing to obtain local proximity features that describe the similarity relationship between network packets. The local proximity features are combined with the original features of the network packet to describe the network packet. The performance of IDS is improved by the increase in effective information in preprocessing.

The motivation to introduce graph representation for network packets preprocessing is as follows. The graph representation learning algorithm has achieved good results in the application of approximate semantic classification of words [12]. An interesting analogy is that we treat a network packet as a word. Then, the problem of distinguishing the class of network packet according to the similarity relationship between network packets can be regarded as the approximate semantic classification problem of words. In the approximate semantic classification of words, similar words tend to appear in similar word neighbors [10]. Therefore, network packets belonging to the same class also have similar local neighbors, where the class consists of DDOS, HttpDos, normal, Brute Force SSH, Infiltrating [13–15]. That is, whether there are similar local neighbors has the ability to characterize the similarity between network packets. Therefore, we believe that the features obtained by preprocessing network packets based on graph representation learning algorithm have the ability to achieve better performance of IDS.

The main contributions of this paper are as follows. (1) In order to improve the performance of the existing unsupervised algorithms of intrusion detection, this paper proposes a packet2vec learning algorithm to extract the local proximity features of network packets. The proposed algorithm based on graph representation by considering the relationship between network packets, and then uses deep Auto-encoder for an intrusion detection system. The system is named by Packet2vec-AE. Compared with the existing preprocessing algorithms for extracting the features of network packets, our proposed packet2vec learning algorithm combines the original features of the network packet and the local proximity features that characterize the similarity between network packets. Therefore, the accuracy of unsupervised intrusion detection is improved. (2) In this algorithm, we construct a relational graph G' by using each packet as a node, calculate the cosine similarity between packets as edges, and then explore the low-order proximity of each packet via the penalty-based random walk in G'. We use the above algorithm as a preprocessing method to enhance the accuracy of unsupervised IDS by retaining the local proximity features of packets maximally. This is the first time in the intrusion detection to

extract features using the graph representation leaning algorithm. (3) In this paper, the penalty is added on the basis of the existing graph representation algorithm node2vec [10].This method can increase the probability of selecting nodes within the local proximity, and the closer the source network packet is, the greater the probability of being selected. Therefore, this method has the ability to extract the local proximity features of the network packet more accurately, and thus more accurately describe the similarity relationship between the network packets. (4) The local proximity features are obtained to accurately characterize the similarity relationship between network packets from the similarity relational graph. The penalty term is used to limit the random walk range of node2vec [10] to k-order proximity. In detail, k is a positive integer that can be customized. This approach has the ability to accurately characterize the neighbor structure of network packets. Experiments have shown that using packet2vec-AE with penalty has the ability to achieve better performance than packet2vec-AE without penalty. In the best case, the accuracy, the detection rate, the precision, and the F_1 of packet2vec-AE are up to 94.7%, 90.9%, 94.3%, and 92.6%, respectively. The proposed algorithm achieves the best performances regarding the accuracy, the precision, and the F_1 exceeding those of the other state-of-the-art algorithms by 11.6%, 11.9% and 8.7%, respectively. In the worst case, the accuracy, the detection rate, the precision, and the F_1 of packet2vec-AE reached 87.4%, 81.1%, 87.3% and 84.1%, respectively. In the worst case, the proposed algorithm achieves the good performances regarding the accuracy, the precision, and the F_1 exceeding those of the other state-of-the-art algorithms by 4.3%, 4.9% and 0.2%, respectively. (5) An empirical formula, i.e., the pruning threshold $\varepsilon \approx$ the mean of weights in $G' - depth' *$ penalty value η, is designed to calculate the approximately optimal penalty value. The experimental results show that the penalty value calculated by this formula has the ability to obtain better intrusion detection performance.

Section 2 describes related work. Section 3 introduces the IDS based on packet preprocess using packet2vec-Autoencoder. Section 4 introduces the experiment. Section 5 discusses the results. Section 6 concludes the paper.

2. Related Work

2.1. Unsupervised Intrusion Detection Techniques

Intrusion detection mainly includes signature-based intrusion detection and anomaly-based intrusion detection. In detail, signature-based intrusion detection is also known as rule-based intrusion detection. Signature-based intrusion detection has a high detection rate for known attacks, but has no ability to detect attack behaviors that are not in the rule base [4]. Anomaly-based intrusion detection is also known as behavior-based intrusion detection. This algorithm has the ability to detect unknown attack behavior [5]. Feature extraction based on accurately preprocessed network packets in anomaly-based intrusion detection is the basis for good performance. At present, it is easy to collect a large amount of unlabeled data in intrusion detection, and it is difficult to obtain a large amount of labeled data [16–18]. Therefore, an unsupervised algorithm for intrusion detection has been introduced.

In recent years, there have been some studies based on unsupervised intrusion detection techniques. In 2007, Liu et al. used a hierarchical PCA model to detect intrusion behavior on the KDD99 dataset [8]. In 2017, Gouveia et al. used RBM to detect intrusion behavior on the ISCX2012 dataset [7]. In 2018, Farahnakian et al. used Deep Auto-encoder to detect intrusion behavior on the KDD99 dataset [9]. The common point of the above studies [7–9] is that the preprocessing ignores the similarity between network packets. Network packets with strong similarity relationship are closer in the similarity relation graph and have similar local proximity embedding [10]. The preprocessing algorithm in the above studies ignores the similarity relationship between network packets, resulting in low accuracy of intrusion detection. Therefore, we introduce a graph representation leaning algorithm to solve the above problem. The network packet is preprocessed by using a graph representation leaning algorithm. In the proposed algorithm, the original features of the network packet are combined

with the local proximity features describing the similarity relationship between the network packets as the input of the neural network. The proposed algorithm adds effective features by considering the similarities between network packets.

2.2. Graph Representation Learning

Graph representation leaning refers to the automatic extraction of proximity features of nodes in a graph [1].This algorithm has achieved effects in the application of approximate semantic classification of words [12], handwritten character classification [19] and so on. The existing graph representation leaning algorithms are mainly the following, such as: Node2vec [10], Deepwalk [20], LINE [21]. However, Deepwalk [5] uses a simple unbiased random walk, which does not control the direction of random walks. Therefore, Deepwalk is easy to sample into the higher-order proximity range, which makes it impossible to accurately describe the local proximity features of the current node; LINE [21] does not have the ability to simultaneously sample first-order proximity and second-order proximity, so the algorithm has limitations; Node2vec [10] uses a random walk based on breadth first search (BFS) or depth first search (DFS) to explore the neighbor of the node. However, random walks based on BFS or DFS are blind searches [22]. In other words, BFS lacks constraints on the range of random walks. Therefore, the random walk has the probability of sampling to the source node's higher-order proximity, which makes it impossible to accurately extract the local proximity features of the source node that describe the similarity relationship between network packets. Therefore, we propose a node2vec with penalty for unsupervised automatic preprocessing. The proposed algorithm limits the range of random walks so that the sampling of random walks only occurs within the low-proximity of the source node. Therefore, the algorithm has the ability to obtain local proximity features that accurately characterize the similarity of network packets.

The literature that we surveyed has not used the unsupervised automatic preprocessing of graph representation learning algorithm with penalty in the field of intrusion detection. In the proposed algorithm, the original features of the network packet are combined with the local proximity features describing the similarity relationship between the network packets as the input of the neural network. The proposed algorithm adds effective features to get better intrusion detection performance.

3. Proposed Intrusion Detection System

3.1. Definition

IDS multi-classification: It refers to the problem of determining that the current network packet belongs to one of multiple types including normal, DDoS, Http DoS, brute force SSH, and infiltrating attacks [23].

Undirected complete graph $G(V, E, W)$**:** It refers to a collection of vertices (or nodes) $V = \{v_1, \ldots, v_n\}$, edges $E = \{e_{ij}\}^n$ and weights $W = \{w_{ij}\}^n$ [24–26].

Relational graph on packets similarity: It refers to treating each packet as a node $v_i \in V$. Calculating the cosine similarity between network packets (v_i, v_j) as the weight $w_{ij} \in W$. The vector of the network packet used to calculate the cosine similarity is the original features of the network packet. The detailed description of the original features of the network packet is in Section 3.3. An undirected complete graph G is constructed using the above method. In order to reduce the use of memory, the edges whose weights are lower than the threshold ε are pruned. G after pruning is called relational graph on packets similarity, expressed as $G'(V, E, W)$.

Pruning threshold: It refers to a value for pruning the edges whose weights are lower than the threshold ε in the graph G'.

Source network packet(alias name: Source node): It refers to the starting network packet for random walk.

Packet embedding: Given a relational graph on packets similarity $G'(V, E, W)$, it refers to a mapping $f : V \rightarrow R^d$. In details, f is the mapping function from node to feature representation.

Automatic preprocessing: It refers to the automatic extraction of features from the original packet and eliminates manual intervention [6].

K-order proximity: It refers to a kind of feature that captures the k-hop relationship between each pair of vertices.

First-order proximity: It refers to the local pairwise proximity between two connected vertices, which captures the direct neighbor relationship between the vertices. For each vertex pair (v_i, v_j), if $(v_i, v_j) \in E$, the first-order proximity between v_i and v_j is w_{ij}; otherwise, the first-order proximity between v_i and v_j is 0 [4]. First-order proximity is equivalent to k in the k-order proximity equal to 1. In *a)* of Figure 1, nodes v_1, v_2, v_3, v_4, v_5 are the first-order proximity of v_i.

Second-order proximity: It refers to capturing a two-hop relationship between each pair of vertices, which describes the proximity of the pair's neighbor structure [5]. Second-order proximity is equivalent to k in k-order proximity equal to 2. In Figure 1a, nodes $v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}$ are the second-order proximity of v_i.

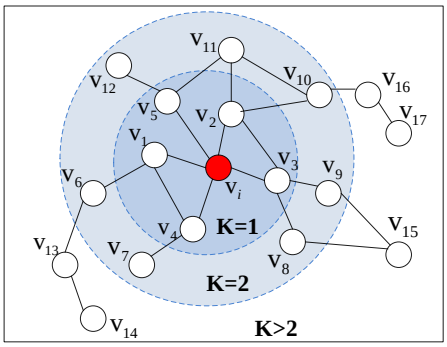
Low-order proximity: It refers to a kind of feature that captures the k-hop relationship between each pair of vertices, where k is lower than 3. It contains first-order proximity and second-order proximity.

High-order proximity: It refers to capturing the k-hop relationship between each pair of vertices, which captures a more global structure [5]. k is greater than or equal to 3. In Figure 1a, nodes $v_{13}, v_{14}, v_{15}, v_{16}, v_{17}$ are high-order proximity of v_i.

Proximity sampling strategy: It refers to sampling the network packets in the neighbor of the source network packet v_i by random walk [10], expressed as S. In Figure 1b, the proximity sampling strategy of the existing algorithm [10] is used for sampling to obtain a proximity sampling list of the source network packet. In Figure 1c, the proximity sampling strategy of the proposed algorithm is used for sampling to obtain a local proximity sample list of source network packets.

Local proximity sample list: It refers to the result of sampling with the proximity sampling strategy S in the low-order proximity of the source network packet v_i, expressed as $N_s(v_i)$. In G', the closer the network packet is, the more similar the local neighbor sample list of the network packet is [10]. Therefore, the local proximity sample list has the ability to accurately describe the similarity relationships between network packets.

Local proximity features: It refers to optimizing the local proximity sample list $N_s(v_i)$ with Skip-gram [12] to obtain continuous d-dimension features. The local proximity sample list $N_s(v_i)$ has the ability to accurately characterize the similarity between network packets. Therefore, local proximity features also have the ability to accurately characterize relationships between network packets.



(a)

Figure 1. *Cont.*

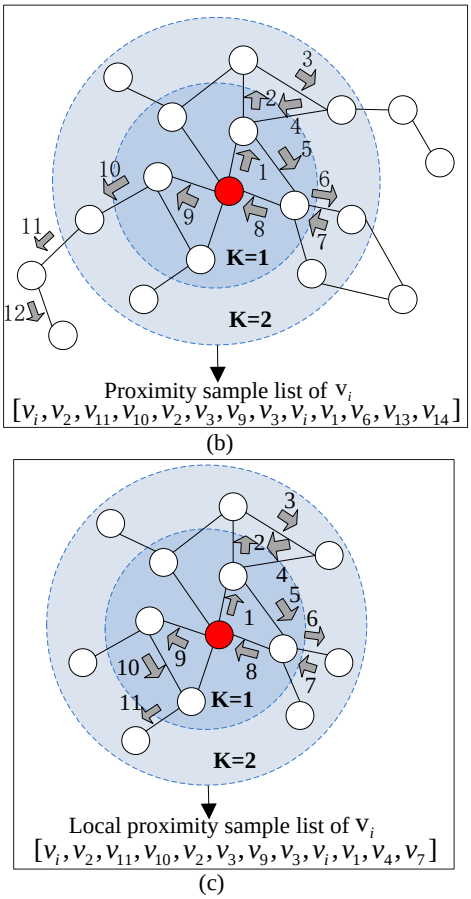


Figure 1. Proximity and comparison based on graph representation. The proximity sampling strategy cannot accurately characterize the local proximity features of the network packet with unlimited proximity. The local proximity sampling strategy has the ability to accurately characterize the local proximity features of the network packet when the sampling range of the source network packet is limited to a low-order proximity. Section 3.4 will introduce the detailed algorithm of graph representation learning for low-order proximity feature extraction. (**a**) Proximity nodes of source network packet v_i. The red node represents the source network packet v_i. (**b**) A proximity sampling strategy is used for random walk to obtain a proximity sample list of the source network packet. The number of the gray arrow indicates the order in which the sampling strategy is used for sampling. (**c**) A local proximity sampling strategy is used to perform a random walk of the constraint range to obtain a local proximity sample list of the source network packet.

3.2. Flowchart

Figure 2 is a flow chart of the algorithm for intrusion detection. Intrusion detection is mainly divided into three parts. In the first part, the originally collected traffic is split into packets. The second part is the automatic preprocessing to obtain the features of the network packet. In this part, we combine the original features of the network packet with the local proximity features of the network packet obtained by packet2vec preprocessing. In the third part, the features obtained by the preprocessing are used as the input of the deep Auto-encoder for intrusion detection.

Data preprocessing is the basis for improving the performance of the entire IDS, because the network packet features obtained by data preprocessing directly affect the final performance of the IDS [6]. Network packets in the pre-processing of existing unsupervised intrusion detection algorithms are generally considered to be independent. The existing algorithm ignores the similarity relationship between the network packets. Therefore, existing algorithms suffer from low intrusion detection accuracy. The higher the similarity of the network packets, the more similar the local proximity features sampled by packet2vec in the relational graph of network packet similarity.

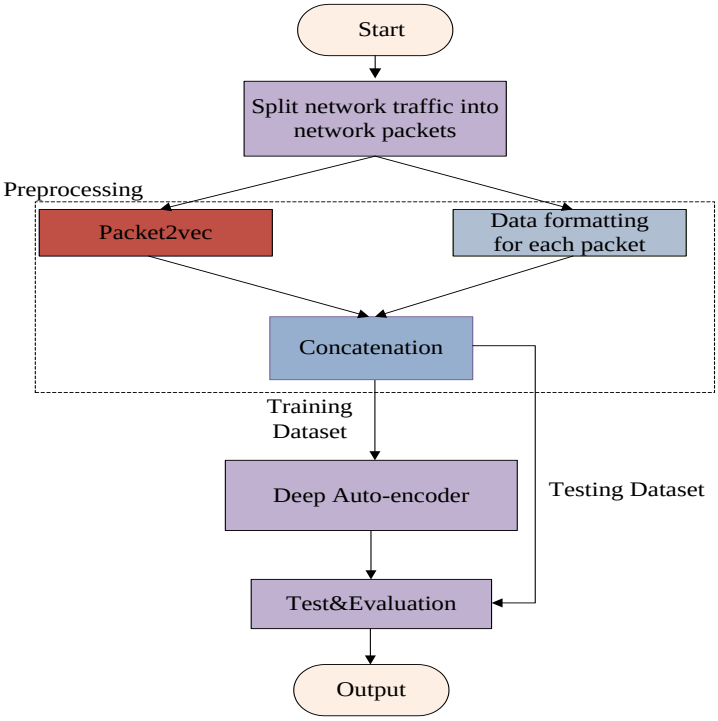


Figure 2. Flowchart of intrusion detection system. Especially, the packet2vec is added to extract low-order proximity feature of network packets in the step of preprocessing to enhance the performance of IDS.

3.3. Unsupervised Preprocessing Based on Packet2vec Learning with Graph Representation

As a preparation step for preprocessing of network packets, we convert the original network traffic into network packet features. Network packets are the basic unit used to determine whether network traffic flow is intrusive. Therefore, we use SplitCap [6] to group interacting source IPs and destination IPs in network traffic into the same network packet [6].

The preprocessing steps for network packets are as follows. Constructing a relational graph on packets similarity $G'(V, E, W)$. The local proximity features in G' obtained by packet2vec is called $v_{i\_packet2vec}$. This step will be detailed in Section 3.4. In addition, we also need to extract the original features $v_{i\_original}$ in the network packet. Finally, the feature I describing the network packet is obtained in combination with $v_{i\_original}$ and $v_{i\_packet2vec}$. For example, if the original features of the network packet are $v_{i\_original} = [0a\ 5c]$, the local proximity features of the network packet obtained by packet2vec are $v_{i\_packet2vec} = [de\ 87]$. Then the features of the network packet obtained after preprocessing are $I = v_{i\_original} + v_{i\_packet2vec} = [0a\ 5c\ de\ 87]$. The original features extraction algorithm of the network packet refers to intercepting the first r bytes of the network packet, and then each byte within [0, 255] in the r bytes obtained from the network packet corresponds to a feature [6]. As shown in Figure 3, the originally traffic is split into packets to be truncated into fixed r bytes. This algorithm of extracting the original features in the network packet is the same as the algorithm of extracting the features of the network packet in [6,27]. The original features extracted by this algorithm have the ability to characterize network packets. Experiments have proved that the algorithm in [6,27], as the original features extraction of this paper, has achieved better results than most of existing algorithms. Algorithm 1 describes the flow of unsupervised preprocessing based on packet2vec learning with graph representation

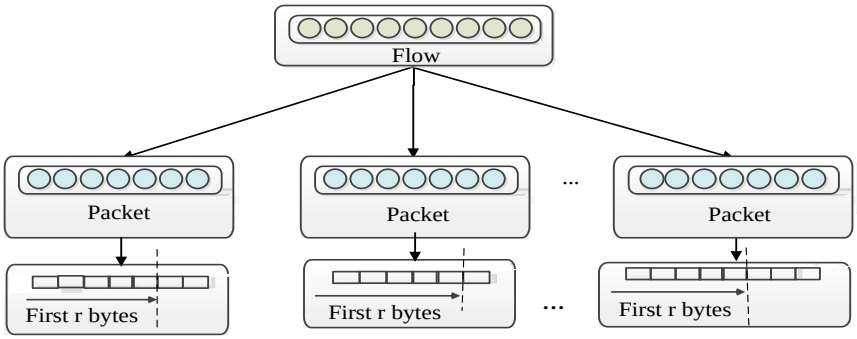


Figure 3. Splitting the originally traffic into packets. The original traffic is split into packets to be truncated into fixed length.

Algorithm 1 Unsupervised preprocessing based on packet2vec learning with graph representation

Input: network traffic flow (f_i).
Output: *Vec* consists of q packets of network traffic, each packet vector $(vec_{i1}, vec_{i2}, \cdots, vec_{ir})$.

Step 1:Split network traffic flow
Using splitcap to split the network traffic flow into q packets $(p_1, p_2, \cdots, p_q)$.
Step 2:Construct relational graph on packets similarity G'
Each network packet is treated as a node v_i. Each byte in the r bytes obtained from the network packet v_i corresponding to a feature, then we get vector $v_i = v_{i\_original} = (vec\_original_{i1}, vec\_original_{i2}, ..., vec\_original_{ir})$.
Calculating the cosine similarity between any two packets (v_i, v_j) as the weight w_{ij} between the two packets. Constructing an undirected complete graph G.
Pruning the edges with similarities below the threshold ε. The graph G after pruning is called relational graph on packets similarity, which is denoted as $G'(V, E, W)$.
Step 3:Extract the local proximity features by using packet2vec learning
Using packet2vec preprocessing to obtain the local proximity feature of each network packet in G', called $v_{i\_packet2vec}$. This part will be described in detail in Algorithm 2.
Step 4:Construct vector as the input of deep auto-encoder
while the information of q packets is not extracted **do**
    Combining $v_{i\_original}$ and $v_{i\_packet2vec}$ to get I. $I = v_{i\_original} + v_{i\_packet2vec} = (vec_{i1}, vec_{i2}, ..., vec_{ir})$, where $v_{i\_original}$ is the original feature of the i^{th} network packet, $v_{i\_packet2vec}$ is the local proximity feature of the i^{th} network packet preprocessed by the packet2vec, and I is the feature of the i^{th} network packet.
end while
Smoothing.
Vec consists of q packets, each packet vector I.
return vector *Vec*.

3.4. Extract the Local Proximity Features by Using Packet2vec Learning

3.4.1. Overview

The proximity sampling strategy S refers to sampling the network packets in the proximity of the source network packet v_i by random walk. We have numbered each network packet. Therefore, the result of sampling by the proximity sampling strategy S is a list N_S of network packet numbers obtained by random walk. The local proximity sampling list $N_S(v_i)$ of a source network packet v_i use the strategy S to sample high-similarity packets within the low-order proximity of v_i. Therefore, we want to perform random walks only within the range of low-order proximity of the source network packet v_i, as shown in c) of Figure 1. However, the existing graph representation learning algorithm has no ability to control the random walk range, which makes it easy to random walk to a high-order

proximity farther away from the source network packet v_i. The random walk of the existing graph representation learning algorithm is shown in b) of Figure 1. Therefore, existing algorithms suffer from inaccurate local proximity sampling list $N_S(\nu_i)$ of source network packet v_i. In detail, the result of the local proximity sampling list $N_S(\nu_i)$ of v_i is the basic of the local proximity features of v_i. Therefore, the local proximity features obtained by existing preprocessing algorithms does not have the ability to accurately describe the similarity relationship between network packets.

In order to control the random walk range of node2vec within the k-order proximity, we introduced Astar to increase the penalty constraint on the random walk of node2vec. The algorithm uses the penalty term to constrain the proximity sampling range of the source network packet v_i to ensure that the random walk is within the k-order proximity, where k is a positive integer and k can be customized. In this paper, the value of k is 2. That is to say, the random walk range of the source network packet v_i is within the low-order proximity. Accordingly, the proposed algorithm that uses Astar to increase the penalty constraint on the random walk range of node2vec is called the packet2vec learning algorithm. In detail, the target of the packet2vec learning algorithm is that each network packet is mapped to the feature $f = V \to \mathbb{R}^d$, where d is the dimension of the features obtained by the mapping, and f is a mapping function. Therefore, we find the mapping function f to map each network packet to obtain d-dimensional features. As a preliminary step of the packet2vec learning algorithm, the relational graph on packets similarity G' is constructed. Each network packet $v_i \in V$ is treated as a node. Two basic steps of the packet2vec learning algorithm are as follows.

First of all, using the proximity sampling strategy S to simulate a random walk process with a length l in the neighbor of v_i to obtain a local proximity sample list $N_S(\nu_i)$. The specific process to obtain the local proximity sampling list $N_S(\nu_i)$ of the source network packet v_i is as follows. (1) The source network packet is $v_i \in V$, and then the proximity sampling strategy S is used to perform penalty-based random walk in the proximity of the source network packet; (2) At each step of the penalty-based random walk, the current weight is updated according to the penalty-based weight update method. The updated weight is the new transition probability n_{vx}; (3) Then select a packet for next step of the penalty-based random walk, which is equivalent to simulating the Alias sampling with time complexity $O(1)$ according to the updated transition probability n_{vx} [10]; (4) The above steps (2) and (3) are continuously repeated until the local proximity sampling list $N_S(\nu_i)$ of length l is obtained. The local proximity sampling list of v_i obtained by the above algorithm is $N_S(\nu_i)$, also known as *walk*. In detail, $N_s(v_i) \subset V$, and the sampling range of penalty-based random walk is not limited to the direct neighbor, but can be sampled by the proximity sampling strategy S within the low-order proximity of the source network packet v_i.

Next, we use Skip-gram [28] to optimize the proximity sample list $N_S(\nu_i)$ to obtain continuous d-dimension features. The specific process is as follows. We seek to optimize the following objective function, which maximizes the log-probability of observing a network proximity $N_S(\nu_i)$ for a network packet v_i conditioned on its feature representation, given by f [10]. The objective function is shown as (1). In particular, the Skip-gram [12] aims to learn continuous feature representations for source network packet v_i by optimizing a proximity preserving likelihood objective. The network packet feature representations are learned by optimizing the objective function using SGD with negative sampling [28]. Finally, we obtain a continuous d-dimensional local proximity feature that accurately characterizes the similarity between network packets to optimize the local neighbor proximity sample list $N_S(\nu_i)$ motivated by node2vec algorithm in [10].

$$\max_f \sum_{u \in V} \log \Pr\left(N_s(v_i) \left| f(v_i)\right.\right), \tag{1}$$

$$\Pr\left(N_s(v_i) \left| f(v_i)\right.\right) = \prod_{n_i \in N_s(v_i)} \Pr\left(n_i | f(v_i)\right), \tag{2}$$

$$\Pr\left(n_i | f(v_i)\right) = \frac{\exp\left(f(n_i) \cdot f(v_i)\right)}{\sum_{v \in V} \exp\left(f(v) \cdot f(v_i)\right)}. \tag{3}$$

Figure 4 is a flow chart of the packet2vec learning algorithm. Algorithm 2 describes in detail the algorithm flow of obtaining the local proximity feature of the source network packet by using packet2vec preprocessing. A description of several key operations involved in Algorithm 2 is as follows.

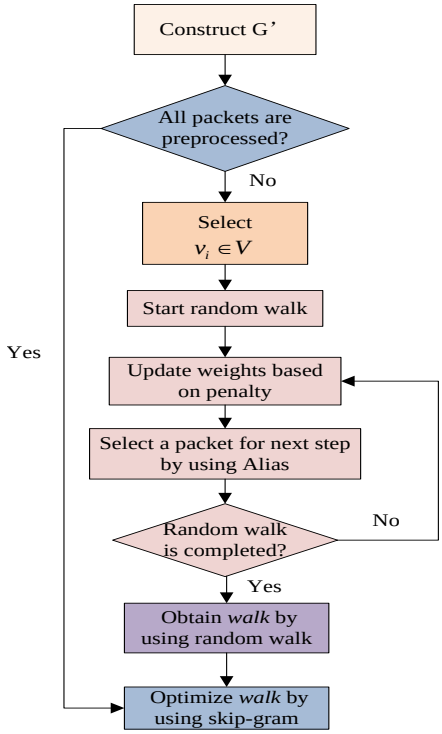


Figure 4. Algorithmic flowchart of packet2vec learner based on graph representation with penalty.

Algorithm 2 Packet2vec learning algorithm

Input: Relational graph on packets similarity $G'(V, E, W)$, Walk length of proximity sampling l, Probability of returning to the previous node p, Probability of moving away from the source node q, Penalty value η.

Output: Local proximity features of i^{th} network packet $v_{i\_packet2vec}$, which is d dimensional.

1: Initialize *walk* to Empty.
2: **for** each node $v_i \in V$ **do**
3: *walk* = **packet2vecWalk** (G', v_i, η, l).
4: **end for**
5: *featur_embedding* = Skip-gram optimization(*walk*).
6: **return** *feature_embedding* $v_{i\_packet2vec}$.

3.4.2. Astar: Penalty-Based Weight Update Method

The traditional method uses BFS random walk to obtain network packets with high similarity to the source network packets. BFS can't limit the range of random walks, and it is easy to cause random walk to high-order proximity, as shown in Figure 1b. Therefore, a penalty-based weight update method is introduced, which is called A star. This method is equivalent to adding a penalty constraint on the BFS, so that the range of random walk is within the k-order proximity of the source network packet. The specific process of the penalty-based weight update method is as follows. First, we calculate the penalty value η according to the empirical formula, i.e., (the pruning threshold $\varepsilon \approx$ the mean of weights in $G' - depth'$ * penalty value η). In detail, we limit the random walk within the k-order proximity of the source network packet, then $depth'$ is equal to k. This article defines the value of $depth'$ to be 2 in our experimentation. The mean of the edges is the sum of the weights of

all edges in the relational graph on packets similarity G' divided by the total number of edges; Next, we calculate the penalty term g_{vx} of the edge according to (5) at each step of the random walk; The sum of the penalty term g_{vx} and the biased weight h_{vx} of the edge is calculated according to (4), which is called the penalty-based weight n_{vx}. Then updates the weight value of the edge in G' according to the penalty-based weight n_{vx}. The graph after the weight update is recorded as G'_cur_weight. Finally, the penalty-based weights of edge below the threshold ε are pruned. Therefore, this method has the ability to control the random walk range within the k-order proximity of the source network packet. Several key operations involved in the penalty-based weight update method are described below.

Biased weight h_{vx}: h_{vx} is a biased weight, and the calculation method is as shown in (6), where w_{vx} is the weight of the edge, and search bias $\alpha_{pq}\left(v_{pre}, x\right)$ has the ability to roughly control the direction of random walk. The calculation method of biased weight is same with that of the transition probability from the current node to the next node in the reference [10].

Search bias $\alpha_{pq}\left(v_{pre}, x\right)$: $\alpha_{pq}\left(v_{pre}, x\right)$ has the ability to roughly control the direction of random walks, such as: approximate DFS, approximate BFS. This does allow us to account for the network structure and guide our search procedure to explore different types of network proximities [10]. This paper mainly samples the local proximity of the source network packet, so the random walk of the approximate BFS is used in this paper. The calculation method of search bias $\alpha_{pq}\left(v_{pre}, x\right)$ is shown in (7), where x is the next network packet, v_{pre} is the previous network packet, $d_{v_{pre,x}}$ is the shortest path between the nodes v_{pre} and x. Therefore, the value of $d_{v_{pre,x}}$ is a value in 0, 1, 2. Figure 5 illustrates the method of search bias $\alpha_{pq}\left(v_{pre}, x\right)$ roughly control the direction of random walks. $\alpha_{pq}\left(v_{pre}, x\right)$ defines two parameters p and q to guide the direction of random walk. Intuitively, parameters p and q control how fast the walk explores and leaves the proximity of starting network packet v_i [10].

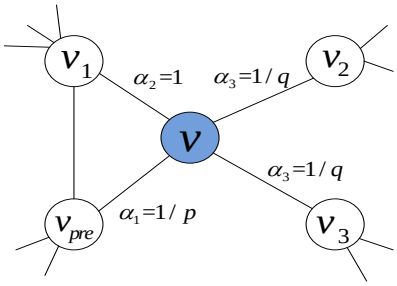


Figure 5. Searching bias $\alpha_{pq}\left(v_{pre}, x\right)$. The walk just transitioned from v_{pre} to v is now evaluating its next step out of node v. And v_{pre},v_1,v_2,v_3 are the nodes that may be reached from node v in the next step. Edge labels indicate searching biases α. It is worth noting that v is the current network packet. v_{pre} is the previous network packet, so searching bias $\alpha_1 = 1/p$. v_1 is connected to v and connected to v_{pre}, so searching bias $\alpha_1 = 1$. v_2 and v_3 are connected to v, while v_2 and v_3 are not connected to v_{pre}, so searching bias $\alpha_3 = 1/q$.

Parameter p: The parameter p controls the possibility of revisiting the previous network packet immediately during the random walk [10]. This article mainly uses random walk of approximate BFS, so the value of p is usually $(p = \max(p, 1))$.

Parameter q: The parameter q controls that random walks tend to access network packets farther away from the network packet v_{pre}. This article mainly uses random walk of approximate BFS, so the value of q is usually $(q = \min(q, 1))$.

If the p value is too large, the random walk may often return to the previous network packet, and it is easy to fall into the local loop search; When the p value is too small or the q value is too large, the random walk is easy to sample to the high-order proximity of the source network packet. From the above analysis, it can be concluded that the random walk with only the biased weight h_{vx} has no ability to accurately characterize the local proximity features of the source network packet. Therefore,

we use the penalty g_{vx} to constrain the range of random walks to obtain local proximity features that have the ability to accurately characterize the similarity of network packets.

Penalty g_{vx}: g_{vx} is the penalty for the next network packet, and the calculation method is as shown in (5), where $depth'$ is the shortest path length from the source network packet to the next network packet. η is the penalty value, so $\eta <= 0$. The role of g_{vx} is to increase the penalty if the next network packet is far away from the source network packet during random walk. The farther the next network packet is from the source network packet, the larger the penalty g_{vx}. The penalty g_{vx} has the ability to control the range of random walk;

Penalty-based weight n_{vx}: n_{vx} is a penalty based weight, also known as the transition probability from the current network packet to the next network packet. The penalty based weight n_{vx} is calculated as shown in (4). Considering a random walk that just traversed edge (v_{pre}, v) and now resides at network packet v [10]. When a random walk requires the selection of a network packet for the next step, the penalty-based weight n_{vx} on the edge (v, x) needs to be evaluated. The penalty based weight n_{vx} is the sum of the penalty g_{vx} and the biased weight h_{vx}, where v is the current network packet. If the penalty term g_{vx} is 0, the packet2vec leaning algorithm is the node2vec [10] leaning algorithm.

$$n_{vx} = g_{vx} + h_{vx}, \tag{4}$$

$$g_{vx} = depth^{'} * \eta, \tag{5}$$

$$h_{vx} = \alpha_{pq}\left(v_{pre}, x\right) \cdot w_{vx}, \tag{6}$$

$$\alpha_{pq}\left(v_{pre}, x\right) = \begin{cases} \frac{1}{p} & \text{if } d_{v_{\text{pre}}x} = 0, \\ 1 & \text{if } d_{v_{\text{pre}}x} = 1, \\ \frac{1}{q} & \text{if } d_{v_{\text{pre}}x} = 2. \end{cases} \tag{7}$$

Node2vec based on penalty constraints has the ability to obtain local proximity features of each network packet more accurately. In the local proximity representation of each network packet, a random walk is used to capture the relationship between network packets. The relational graph on packets similarity G' is transformed into a set of network packet lists by random walk. The frequency of occurrence of the network packet pairs in the set measures the structural distance between the network packet pairs [10]. In detail, the closer the network packet is, the higher the similarity of the network packet. Algorithm 3 details the weight update strategy of the similarity relationship graph.

Algorithm 3 Penalty-based weight update for similarity relation graph (Abbreviated as PBWeight)

Input: Relational graph on packets similarity $G'(V, E, W)$, Probability of returning to the previous node p, Probability of moving away from the source node q, Penalty value η, Current node cur, Shortest path length between each pair of nodes $dijkstra\_path\_length$, proximity nodes of the current node v_{cur}, Pruning threshold ε.

Output: Weight after punishment cur_weight, Relational graph of network packet similarity after punishment G'_cur_weight.

G'_cur_weight = Deepcopy(G').
for $proximity\_node$ in v_{cur} **do**
    g(cur)= $dijkstra\_path\_length[cur][proximity\_node] * \eta$
    h(cur)= $W * \alpha_{pq}$
    cur_weight = g(cur) + h(cur)
    if $cur\_weight > \varepsilon$ **then**
        Update graph G'_cur_weight according to the value of cur_weight
    end if
end for
return $cur\_weight, G'\_cur\_weight$.

3.4.3. Packet2vecwalk: Random Walk with Penalty for Packet2vec Learning Algorithm

We consider the proximity of the source network packet from the similarity relational graph G' as a local search problem. We propose a flexible proximity sampling strategy based on penalty for random walk, which controls the range of random walk. The proposed algorithm uses random walk similar to BFS.

Random walk: The source network packet is v_i, and the length of the random walk we need to simulate is l. Our goal is to generate a local proximity sample set $N_S(v_i)$ of the source network packet v_i. Assume that random walks are started from the source network packet $c_0 = v_i$, and the m^{th} network packet in the random walk is c_m. In detail, the network packet c_m is generated by the following distribution [10].

$$\mathrm{P}(c_m = x | c_{m-1} = v) = \begin{cases} \frac{n_{vx}}{z} & \text{if } (v,x) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

where n_{vx} is the weight based on the penalty. n_{vx} is the weight between the nodes v and x after the update based on the penalty weight. Z is the normalizing constant [10]. E is a collection of edges. Algorithm 4 describes the random walk with penalty for packet2vec learning algorithm.

Algorithm 4 Packet2vecWalk: Random walk with penalty for packet2vec learning

Input: Relational graph on packets similarity $G'(V, E, W)$, Start node v_i, Penalty value η, Walk length of proximity sampling l.

Output: Local proximity features of source network packet v_i obtained by packet2vec *walk*.

Initialize *walk* to $[v_i]$.
dijkstra_path_length = Dijstra(G).
for *walk_iter* = 1 to l **do**
    cur_weight , G'*_cur_weight* = **PBWeight** $(G', p, q, \eta, cur, \boldsymbol{l}, dijkstra\_path\_length, v_{cur}, \varepsilon)$.
    v_{cur} = GetProximities(cur, G')
    s = AliasSample(v_{cur}, *cur_weight*)
    Append s to *walk*
end for
return *walk*.

3.5. Auto-Encoder for Intrusion Detection

We use the preprocessed network packet features as input to the deep auto-encoder intrusion detection. Deep auto-encoder consists of two parts, encoder and decoder. Encoder compresses the raw data into a low-dimensional representation. Decoder reconstructs the low-dimensional representation of the encoder compression. Algorithm 5 describes the flow of Auto-encoder based on on packet2vec learning for intrusion detection

Encoder: $X = [x_1, x_2, \cdots, x_n]^T$ is the input data vector of encoder, $X^{'} = [x_1^{'}, x_2^{'}, \cdots, x_m^{'}]^T$ is the output data vector of encoder. It is worth noting that n is larger than m. Encoder uses *tanh* as the activation function.

$$X' = f(x) = \tanh(WX + b). \tag{9}$$

where W represents the encoder weight matrix with size $m * n$ and b is a bias vector of dimensionality m.

Decoder: The decoder function maps the hidden representation X' to a reconstruction $\mathrm{Y} = [y_1, y_2, \ldots, y_n]$. Decoder uses *tanh* as the activation function.

$$Y = g(x') = \tanh(W'X' + b'). \tag{10}$$

where W' represents the decoder weight matrix with size $n \times m$ and b' is a bias vector of dimensionality n.

Object function: The training goal of deep auto-encoder is to minimize the error between input X and output Y.

$$\mathcal{L}(\theta) = \sum_{x \in D} L(x, g(f(x))), \tag{11}$$

$$\mathcal{L}(x, y) = \|x - y\|^2. \tag{12}$$

In addition, we find the optimal parameter θ through the objective function.

$$\theta = \{W, W', b, b'\} = \arg_\theta \min L(x, y). \tag{13}$$

Algorithm 5 Auto-encoder based-on packet2vec learning for intrusion detection

Input: vector *Vec*, which contains q network packets in the network traffic, each network packet is a vector (vec_{i1},vec_{i2},$\cdots$,vec_{ir}).
Output: Evaluate result in test dataset.

Step 1:Create auto-encoder model
Add the 1^{st} **encoder** layer of l_1 units whose activation function is *tanh*.
Add the 2^{nd} **encoder** layer of l_2 units whose activation function is *tanh*.
Add the 3^{rd} **dense** layer of l_3 units.
Add the 4^{th} **decoder** layer of l_4 units whose activation function is *tanh*.
Add the 5^{th} **decoder** layer of l_5 units whose activation function is *tanh*.
Step 3: Train model
while early stop condition is not met **do**
    while training dataset is not empty **do**
        Update weights and bias using *adadelta* gradient descent optimization algorithm.
    end while
end while
Step 4:Test model
Test fine-tuned hyper-parameters with test dataset.
return Evaluate result in test dataset.

4. Evaluation

In this section, the performance of the network intrusion detection algorithm using packet2vec-AE is evaluated based on the ISCX 2012 intrusion detection data set. The purpose of the experiment is as follows.

4.1. Dataset

Most of the existing network intrusion detection data sets are based on manual experience to extract network packet features [23], such as NSL-KDD [17], KDD CUP 1999 [29], and Kyoto2009 [30]. The datasets of the existing raw network packets are ISCX2012 [27] and DAPAR1998 [31–33]. The attacks in ISCX2012 are relatively new [6]. Therefore, we used the ISCX2012 dataset for experiments. Table 1 is a description of the ISCX2012 data set. The data set contains 7 days of traffic data, including normal traffic and four types of attack traffic, such as brute force SSH, DDoS, Http DoS, and infiltrating. Table 2 shows the statistics of ISCX 2012. It can be seen from Table 2 that ISCX 2012 contains a small amount of attack data, so the data set is unbalanced. To solve this problem, we resampled [6] the data set. That is, we undersample [6] the normal type of data, and we oversample [6] the data of the four types of attacks. Table 3 is the data set after resampling. It can be seen from Table 3 that the data set after resampling is balanced.

Table 1. Data Description of the ISCX 2012 Dataset [6].

Date	Data Description	Data Size
11 June	Normal	16.1 GB
12 June	Normal, Brute Force SSH	4.22 GB
13 June	Normal, Infiltrating	3.95 GB
14 June	Normal, HttpDoS	23.4 GB
15 June	Normal, DDOS	23.4 GB
16 June	Normal	17.6 GB
17 June	Normal, Brute Force SSH	12.3 GB

Table 2. ISCX 2012 Dataset [27].

ISCX2012	Train Set		Test Set	
	Count	Percentage	Count	Percentage
Normal	890,726	97.27%	593,811	97.27%
Brute Force SSH	4197	0.46%	2785	0.46%
Infiltrating	6027	0.66%	4017	0.66%
Http DoS	2090	0.23%	1392	0.23%
DDOS	12,673	1.38%	8448	1.38%
Total	915,695		610,453	

Table 3. Statistics after ISCX2012 Balanced Processing.

Dataset	Train Set		Test Set	
	Count	Percentage	Count	Percentage
Normal	136,653	20.0%	45,564	20.0%
Brute Force SSH	136,653	20.0%	45,564	20.0%
Infiltrating	136,653	20.0%	45,564	20.0%
Http DoS	136,653	20.0%	45,564	20.0%
DDOS	136,653	20.0%	45,564	20.0%
Total	683,265	100.0%	227,820	100.0%

4.2. Evaluation Metrics

This paper uses accuracy (ACC), detection rate (DR), precision(P), and F_1 as evaluation indicators [34]. The accuracy rate is an indicator that describes the correctness of the intrusion detection algorithm to detect whether there is an intrusion. The detection rate is used to measure the detection performance of the intrusion detection system. The precision refers to the ratio of the number of positive samples that are actually predicted to positive samples to the number of positive samples predicted by the intrusion detection system. The F1 is the harmonic mean of the precision and detection rates. The formulas are as follows. The meanings of True Positive (TP), False Positive (FP), False Negative (FN), and True Negative (TN) [35,36] are shown in Table 4.

$$Accuracy(ACC) = \frac{TP + TN}{TP + FP + FN + TN}, \tag{14}$$

$$DetectionRate(DR) = \frac{TP}{TP + FN}, \tag{15}$$

$$Precision(P) = \frac{TP}{TP + FP}, \tag{16}$$

$$F_1 = 2 \cdot \frac{\mathrm{P} \cdot \mathrm{DR}}{\mathrm{P} + \mathrm{DR}}. \tag{17}$$

Table 4. Cofusion Matrix.

	Actual Class: True	**Actual Class: False**
Predicted Class: True	TP	FP
Predicted Class: False	FN	TN

4.3. Two Ways to Preprocess Network Packets

There are two ways to preprocess network packets. In method (a), the original features $v_{i\_original}$ of the i^{th} network packet is combined with the local proximity features $v_{i\_node2vec}$ of the i^{th} network packet obtained by the packet2vec without penalty. In detail, the local proximity features obtained by packet2vec without penalty, i.e., $g(n) = 0$ in (2), is equivalent to the local proximity features obtained by node2vec. The features obtained by preprocessing the i^{th} network packet is $I = v_{i\_original} + v_{i\_node2vec}$; In method (b), the original features $v_{i\_original}$ of the i^{th} network packet is combined with the local proximity features $v_{i\_packet2vec}$ of the i^{th} network packet obtained by the packet2vec with penalty. The features obtained by preprocessing the i^{th} network packet is $I = v_{i\_original} + v_{i\_packet2vec}$. From Table 5, it can be concluded that the optimal accuracy and detection rate of the method (b) are 7.3% and 9.8% higher than the optimal accuracy and detection rate of the method (a), respectively. As can be seen from Table 5, when the length of the random walk is 1000, the accuracy and detection rate of the method (a) are greatly reduced, and the accuracy of the method (b) is still as high as 90.1%. One possible explanation is that the lack of penalty constraints for method (a) may result in the inability to control the random walk range when local proximity sampling. Therefore, the method (a) causes random walks to high-order proximity that are farther away from the source network packet. Eventually, the local proximity sampling of the source network packet is inaccurate, and the features obtained by the preprocessing cannot accurately describe the network packet. Therefore, in the following we use method (b) for data preprocessing. In addition, it is worth noting that the comparison and selection of the length of the random walk is detailed in Part E of Section 4.

Table 5. Comparison of IDS performance of Node2vec and Packet2vec preprocessed under different random walk lengths (%).

Length *l*	**Preprocessing Algorithm**							
	(a) Preprocessed with Node2vec				**(b) Preprocessed with Packet2vec**			
	ACC	**DR**	**Precision**	**F_1**	**ACC**	**DR**	**Precision**	**F_1**
10	73.8	73.7	75.2	74.4	89.8	87.9	88.0	87.9
80	**87.4**	**81.1**	**87.3**	**84.1**	**94.7**	**90.9**	**94.3**	**92.6**
1000	65.8	63.5	67.9	65.6	90.1	87.3	88.7	88.0

4.4. The Effectivess of Penalty in Packet2vec Preprocessing by Autoencoder

This section discusses the impact of penalty values η on the performance of IDS in Packet2vec-AE. Different penalty values η may affect the random walk range of the source network packet, and ultimately affect the accuracy of the local proximity features of the source network packet. Therefore, it is necessary to evaluate the impact of different penalty values η in Packet2vec-AE on IDS performance based on experiments. The IDS performance of the Packet2vec-AE penalty values of -0.01, -0.03, -0.05, and -0.1 was evaluated. Table 6 is a statistical value of the similarity relationship between network packets in G'. The statistical value shows that the average of the weights in G' is 0.93. Table 7 shows the IDS performance with different penalty values in Packet2vec-AE. It can be concluded from Table 7 that the packet2vec-AE based IDS achieves optimal performance when the penalty value is -0.03. When the penalty values are -0.01 and -0.1, the performance of the IDS is relatively poor. Figure 6 shows the Packet2vec-AE visualizational results for different penalty values. It can be concluded from Figure 6 that the IDS can accurately distinguish DDoS attacks, and it is difficult to accurately distinguish the other four types of network packets, when the penalty values are -0.01 and -0.1; When the penalty value is -0.05, the IDS has the ability to distinguish the following types

of network packets more accurately, including normal, Brute Force SSH attacks and DDoS attacks; When the penalty value is -0.03, the IDS has the ability to distinguish five class of network packets more accurately.

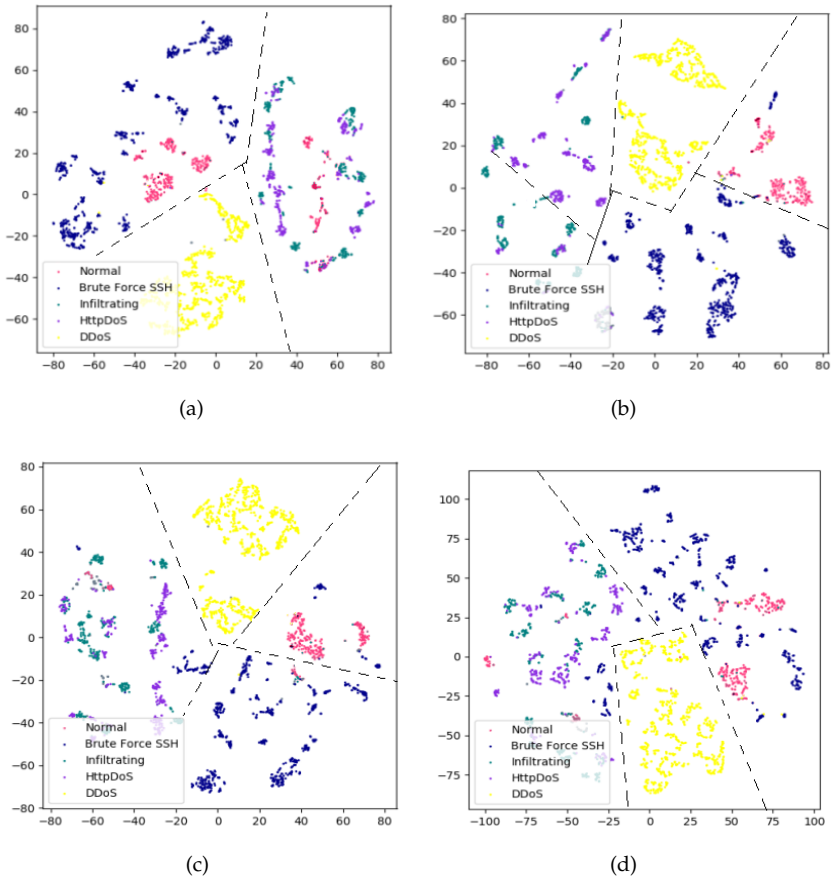


Figure 6. Packet2vec-AE visualizational results with different penalty values. Gray points are normal network packets; The yellow point is the network packet containing the DDoS attack; The blue point is the network packet containing the Brute Force SSH attack; The purple point is the network packet containing the Http DoS attack; The green point is the network packet containing the Infiltrating attack. (**a**) Penalty value $\eta = -0.01$. (**b**) Penalty value $\eta = -0.03$. (**c**) Penalty value $\eta = -0.05$. (**d**) Penalty value $\eta = -0.1$.

One possible explanation is that the penalty value η determines the range of local proximity samples for network packets. Experiment shows that the penalty value calculated by the empirical formula, i.e., (the pruning threshold $\varepsilon \approx$ the mean of weights in $G' - depth'$ * penalty value η), can achieve the optimal effect. In this experiment, the value of the pruning threshold ε is 0.86. The random walk range of the source network packet is within the low-order proximity, that is, the maximum value of $depth'$ is 2. It uses an empirical formula, i.e., (the pruning threshold $\varepsilon \approx$ the mean of weights in $G' - depth'$ * penalty value η), for approximate calculation. When the penalty value is -0.03, the Packet2vec-AE algorithm achieves optimal performance. At the same time, the experimental results in Table 7 and Figure 6 show that the Packet2vec-AE algorithm achieves optimal performance when the penalty value is -0.03. In addition, when the penalty value is -0.1, the penalty for the weight is too large, which leads to too much constraint on the local proximity sampling range. Therefore, an excessive penalty value results in random walk within the first-order proximity of the source network packet when sampling local proximity features of the network packet. The sampling range is too small, so the obtained local proximity features have no ability to accurately describe the similarity relationship between the network packets. When the penalty value is -0.01, the penalty

for the weight is too small, which leads to insufficient constraint on the sampling range of the local proximity. When sampling the local proximity features of the network packet, there is a large possibility of random walk to high-order proximity farther from the source network packet. The sampling range is too large, so the obtained local proximity features do not have the ability to accurately describe the similarity relationship between network packets.

Table 6. Statistical value of the weight in the G'.

Weight in G'	Value
Mean	0.93
Max	0.99
Min	0.86

Table 7. Compare the impact of different penalty values on the performance of Packet2vec-AE (%).

Penalty η	ACC	DR	Precision	F_1
−0.01	89.8	87.9	88.0	87.9
−0.03	**94.7**	**90.9**	**94.3**	**92.6**
−0.05	93.8	89.2	92.6	90.9
−0.1	90.2	87.3	88.7	88.0

4.5. Influence of Packet2vec-AE Hyper-Parameters

Different penalty values η in the similarity relationship of network packets have different effects on the performance of IDS. In addition, other parameters have different effects on the performance of IDS, such as optimizer of auto-encoder, the pruning threshold, the length of random walk. Therefore, these parameters are also adjustable. Table 8 shows the performance of the auto-encoder (AE) in packet2vec-AE with different optimizers. From this table, we have found that *adadelta* as the optimizer for AE has the ability to obtain better performance of IDS. Table 9 shows the effect of different random walk lengths of packet2vec on the performance of IDS. From Table 9, we conclude that the packet2vec-AE algorithm has the ability to achieve optimal performance of IDS when the length of the random walk is 60 or 80. When the random walk length is 10 or 100, the performance of the packet2vec-AE algorithm is relatively poor. One possible explanation is that random walks only sample ten proximity packets of the source network packet when the random walk length is 10. The number of network packets sampled by random walks is small, so the features obtained by random walk do not have the ability to accurately describe the local proximity of the source network packet. Therefore, when the random walk length is 10, the performance of the IDS is not good. When the random walk length is 100, the proximity features obtained by random walk sampling is too redundant. Therefore, when the random walk length is 60 or 80, the features of the local proximity of the source network packet can be accurately characterized. Table 10 shows the architectural parameters of the auto-encoder in the packet2vec-AE algorithm. Table 11 shows a list of parameters for packet2vec in packet2vec-AE. Table 12 shows a list of auto-encoder hyper-parameters and its optimizer in packet2vec-AE. The parameter values given in Tables 11 and 12 are the optimal results obtained by experiments. The hardware used in the experiments are presented below. The configuration of the experimental environment is shown in Table 13. In addition, we used a 12 GHz NVIDIA Tesla K40m GPU as an accelerator.

Table 8. Compare the impact of optimizer on the performance of Packet2vec-AE (%).

Optimizer	ACC	DR	Precision	F_1
adam	90.5	85.9	88.9	87.4
adadelta	**94.7**	**90.9**	**94.3**	**92.6**
SGD	92.9	88.3	92.6	90.4

Table 9. Compare the effects of random walks of different lengths on the performance of Packet2vec-AE (%).

Length l	ACC	DR	Precision	F_1
10	89.8	87.9	88.0	87.9
60	**94.7**	**90.9**	**94.3**	**92.6**
80	**94.7**	**90.9**	**94.3**	**92.6**
100	90.2	87.3	88.7	88.0

Table 10. Architecture Parameters of auto-encoder in the packet2vec-AE algorithm.

Layer	Type	Filter/Neuron
1	encoder+tanh	160
2	encoder+tanh	64
3	encoder+tanh	32
4	dense	10
5	decoder+tanh	32
6	decoder+tanh	64
7	decoder+tanh	160

Table 11. List of Parameters for Packet2vec in Packet2vec-AE.

Hyper-Parameters	Value
probability of returning to the previous node p	2
probability of moving away from the source node q	0.2
penalty values η	-0.3
pruning threshold ε	0.86
truncating network packets lengths	100
length of random work l	60

Table 12. List of Auto-encoder Hyper-parameters and Its Optimizer in Packet2vec-AE.

Hyper-Parameters	Value
Optimizer	*adadelta*
Learning Rate	0.25
Activation function	*tanh*

Table 13. Experimental Enviroment Configuration.

Item	Configuration
Operate System	Linux 3.19.0-25-generic #26 14.04.1-Ubuntu
Hardware	DELL R720,CPU is 16 core Xeon E5-2680
Configuration	2.7GHz,16GB
Python version	Anaconda 2.7

4.6. Comparison with the Latest Techniques

The researchers proposed some unsupervised intrusion detection algorithms such as RBM [7], PCA [8] and deep auto-encoder [9]. Those algorithms usually ignore the similarity relationship between network packets when preprocessing network packet extraction features. The lack of information analyzed by those algorithms results in relatively low accuracy. The experiment compares the performance of the proposed algorithm with the existing algorithms. Table 14 is a comparison of the performance of the various algorithms. Deep auto-encoder preforms better than PCA and RBM in Table 14, so we use deep auto-encoder to implement the proposal for further improvement. It can be seen from Table 14 that the proposed packet2vec-AE algorithm achieves the best performances regarding the accuracy exceeding those of the other state-of-the-art algorithms by 11.6%. The proposed packet2vec-AE algorithm achieves the best performances regarding the precision exceeding those of the other state-of-the-art algorithms by 11.9%. The detection rate was only worse than that of the best algorithm and ranks second among all five algorithms. We used the harmonic mean F_1 to comprehensively assess the detection rate and precision. The proposed packet2vec-AE algorithm

achieves the best performances regarding the F_1 exceeding those of the other state-of-the-art algorithms by 8.7%. The proposed node2vec-AE algorithm achieves the good performances regarding the accuracy, the precision, and the F_1 exceeding those of the other state-of-the-art algorithms by 4.3%, 4.9%, and 0.2%, respectively. We consider node2vec-AE as packet2vec-AE without penalty, so node2vec-AE is considered to be the worst case of the packet2vec-AE algorithm. Figure 7 is a visualization of unsupervised intrusion detection algorithms. It can be seen from Figure 7 that the performance of the PCA algorithm is the worst. The PCA algorithm does not have the ability to accurately distinguish between five different types of network packets. Deep auto-encoder has the ability to accurately distinguish DDoS attacks, but it is difficult to accurately distinguish the other four types of network packets. The proposed node2vec-AE algorithm has the ability to accurately distinguish between several types of network packets, including normal, Brute Force SSH attacks and DDoS attacks. The proposed packet2vec-AE algorithm has the ability to accurately distinguish between five types of network packets. One possible explanation is that existing unsupervised intrusion detection algorithms only use the original features of the network packet as input. The features obtained by packet2vec-AE and node2vec-AE preprocessing include the original features of the network packet, and also include the local proximity features that characterize the similarity between network packets. Due to the increase of effective information in the preprocessed data, the accuracy of the proposed algorithm is higher than the existing three intrusion detection algorithms.

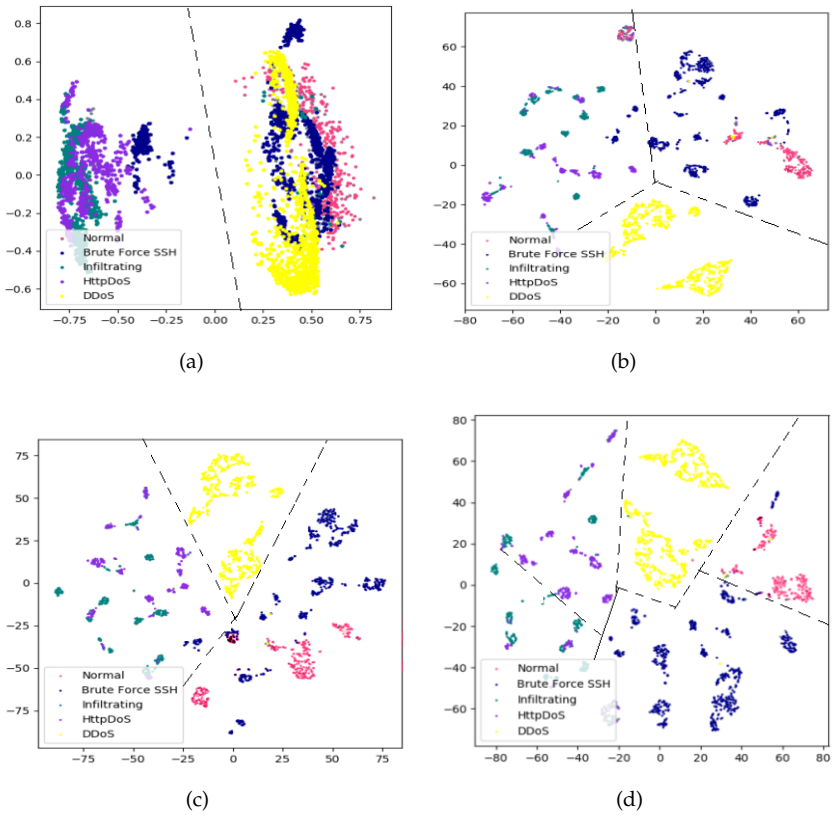


Figure 7. Visualization of unsupervised intrusion detection algorithms. (**a**) PCA algorithm. (**b**) Deep Auto-encoder algorithm. (**c**) Proposed node2vec-AE algorithm. (**d**) Proposed packet2vec-AE algorithm.

From Table 14 and Figure 7, we can see that the accuracy and detection rate of node2vec-AE is slightly lower than packet2vec-AE. One possible explanation is that the node2vec-AE algorithm does not constrain the local proximity sampling range of random walk. Therefore, the node2vec-AE algorithm has a high probability of random walk to high-order proximity far from the source network packet. Finally, the proximity sampling of the source network packet does not have the ability to

accurately characterize the local proximity. However, the accuracy of the node2vec-AE algorithm is higher than that of RBM, PCA, and deep auto-encoder. This shows that combining the original features of the network packet with the local proximity features that describe the similarity between network packets has the ability to achieve better intrusion detection performance. It also proves the effectiveness of our proposed algorithm.

Table 14 contains some supervised algorithms, such as SVM-IDS 2017 [37], J48-IDS 2017 [38], and C4.5-IDS 2016 [39]. The performance of the SVM-IDS 2017 [37], J48-IDS 2017 [38], and C4.5-IDS 2016 [39] algorithms is for reference only. The algorithm we propose does not require performance comparisons with these three algorithms. The algorithm we propose is unsupervised, but the three algorithms are supervised. So this comparison is unfair. In intrusion detection, labels are difficult to obtain. The advantage of an unsupervised algorithm is that it can perform intrusion detection on all network traffic without being restricted by labels.

For training and testing time, all of my experiments were able to run in 24 h under the server configuration shown in Table 13. The experiment of obtaining the local proximity features of the network packet using the Packet2vec algorithm can be completed in 16 G memory within 12 h. The experimental time for intrusion detection using the extracted network packet characteristics as input to Deep Auto-encoder is within 1 h. The overall operating time is within an acceptable range. The ISCX 2012 dataset appeared later. Therefore, we could not find enough literature on training, testing time, and memory size, and we were not able to evaluate it [6].

Table 14. Comparison with Other Published Algorithms (%).

	Algorithm	Accuracy	DR	Precision	F_1
Supervised algorithms	SVM-IDS 2017 [37]	NA	60	59.2	59.28
	J48-IDS 2017 [38]	NA	90.64	86.4	88.14
	C4.5-IDS 2016 [38]	NA	76.4	78.1	NA
Unsupervised algorithms	PCA-IDS 2007 [8]	81.1	78.7	78.9	78.8
	RBM-IDS 2017 [7]	78.6	**96.0**	70.9	81.6
	Deep AE-IDS 2018 [9]	83.1	85.4	82.4	83.9
	Node2vec -AE (proposal)	**87.4**	81.1	**87.3**	**84.1**
	Packet2vec -AE (proposal)	**94.7**	**90.9**	**94.3**	**92.6**

5. Discussion

In this paper, the packet2vec leaning algorithm is used to obtain the local proximity features that accurately describe the similarity relationship of network packets. Next, the features extracted by the packet2vec leaning algorithm are combined with the original features of the network packet to be used as input of the deep auto-encoder for intrusion detection.

This article focuses on the impact of similarity relationships between network packets on intrusion detection performance. Future work will further explore the features of network packets obtained by preprocessing. Network packets will be analyzed from multiple dimensions such as timing [40–42] and protocol type [43–45].

This paper only uses the network packet in the train set to construct a relational graph on packet similarity G'. When extracting local proximity features of the network packet in the test set, it is not necessary to add the network packet in the test set as a node to G'. It is only necessary to calculate the similarity between the current network packet and each network packet in G'. Then the proposed packet2vec leaning algorithm is used to extract the local proximity features of the current network packet. When this algorithm is used to detect a new network packet, there is no need to update G' or re-train. In other words, network packets in the test set do not have to be added to G'. Therefore, the algorithm can save time and memory. In addition, our proposed algorithm has generalization effectiveness for network packets outside the train set, increasing the scalability of the proposed model to the number of test set samples.

In this paper, we use penalty terms to limit the range of random walks, which causes random walks to sample only network packets in the k-order proximity of the source network packet. The advantage of the above method is that we can customize the value of k according to the needs.The empirical formula, i.e., the pruning threshold $\varepsilon \approx$ the mean of weights in $G' - depth' *$ penalty value η, is designed to calculate the approximately optimal penalty value. The maximum value that $depth'$ can be taken in this empirical formula is k. In addition, we use ($depth'$ * penalty value η) for weight penalty in the empirical formula. In the weight penalty, the closer the source network packet is, the smaller the penalty for the network packet. That is to say, the method has a high probability of preferentially sampling network packets with high similarity to the source network packet.

In addition, the local proximity sample list of the source network packets obtained by random walk has the ability to capture the structural relationship and distance relationship between network packets. The source network packet local proximity sample list can measure the distance between network packet pairs based on the frequency of network packet pairs [10]. If the local neighbor sample list of the source network packet is obtained in other ways (for example, all first-order proximity nodes and second-order proximity nodes of the source network packet are directly used to form a local proximity list), the method does not have the ability to obtain structure and distance information between the network packets. Therefore, it is very meaningful to capture the local proximity nodes of the source network packets in a random walk.

In addition, this article focuses on the relationship between network packets and extracts the features of network packets. In the future work, network flows and network packets will be analyzed hierarchically [46–49].

We analyzed the sensitivity of the proposed algorithm. From Table 14, we can draw the following conclusions. In the best case, the accuracy, the detection rate, the precision, and the F_1 of packet2vec-AE are up to 94.7%, 90.9%, 94.3%, and 92.6%, respectively. In the worst case, the accuracy, the detection rate, the precision, and the F_1 of packet2vec-AE reached 87.4%, 81.1%, 87.3%, and 84.1%, respectively. In the worst case, the proposed algorithm achieves the good performances regarding the accuracy, the precision, and the F_1 exceeding those of the other state-of-the-art algorithms. Therefore, the proposed algorithm is still superior to the latest algorithms available even if the selected parameters are not suitable. From the experimental results of Tables 8 and 9, it can be concluded that fluctuation range of the accuracy, the detection rate, the precision, and the F_1 is not large regardless of how these parameters are changed. In other words, the experimental results are not sensitive to the parameters. Therefore, we have reason to believe that the proposed algorithm has the potential to be extended to other data sets. In future work, we will apply this algorithm to other data sets.

In order to highlight the novelty and contribution of this paper, we compare the similarities and differences between our algorithm and existing algorithms in Tables 15 and 16. Table 15 compares the similarities and differences between the latest three unsupervised IDS algorithms and our proposed algorithms. Most existing studies on unsupervised IDS preprocessing ignore the relationship among packets. As a result, the performance of existing unsupervised IDS is not high. According to homophily hypothesis, the local proximity structure in the similarity relational graph has similar embedding after preprocessing. Our proposed algorithm pre-processing combines the local proximity feature of the network packet with the original features of the network packet as input to the Deep Auto-Encoder. Our proposed algorithm is equivalent to the use of local proximity features of network packets to enhance the original features of network packets. From Table 15, we can conclude that the performance of the existing PCA-IDS 2017 [8] and RBM-IDS 2017 [7] depends on the features of manual experience extraction. Our proposed algorithm uses graph representation learning for automatic preprocessing. Therefore, our proposed algorithm is suitable for raw network traffic data. Table 16 compares the latest graph representation of the similarities and differences between the learning algorithm and our proposed algorithm. Deepwalk [20] and Node2vec [10] are easy to sample into the higher-order proximity range, which make it impossible to accurately describe the low proximity features of the current node; LINE [21] does not have the ability to simultaneously sample first-order proximity and

second-order proximity, so the algorithm has limitations; Our proposed algorithm has the ability to extract first-order proximity, second-order proximity, and low-order proximity. This is the first time in the intrusion detection to extract features using the graph representation leaning algorithm.

We have increased the analysis of the effect of random walk length on the running time cost of the algorithm. Intuitively, the length of the random walk is proportional to the runtime overhead of the proposed algorithm. The length of a random walk is the number of times the next node needs to be selected during a random walk. The transition probability needs to be calculated each time the next node is selected during a random walk.

Future research work mainly considers two aspects, including intrusion detection [50,51] with a small number of network packet labels [52–54] and intrusion detection for unknown malicious traffic [55]. The purpose of intrusion detection with only a small number of network packet labels is to build a security baseline based on the full use of existing small amounts of label data. Intrusion detection of unknown malicious traffic is very important in practical applications. We will continue to research the application of deep neural networks in the IDS field with the hope of further improving the IDS performance.

Table 15. Comparison with Other Published Unsupervised Intrusion Detection Algorithms (%).

Algorithms	Raw Traffic Adoptation	Relational Features after Preprocessing	Automatic Feature Extraction	Manual Experience to Extract Features
PCA-IDS 2007 [8]				✓
RBM-IDS 2017 [7]	✓			✓
Deep AE-IDS 2018 [9]				✓
Node2vec -AE (proposal)	✓	✓	✓	
Packet2vec -AE (proposal)	✓	✓	✓	

Table 16. Comparison with Other Published Graph Representation Learning Algorithms (%).

Algorithms	First-Order Proximity	Second-Order Proximity	Low-Order Proximity	Usability for IDS	Usability for Network Packet Feature Extraction
Deepwalk [20]		✓			
LINE [21]	✓	✓			
Node2vec [10]		✓			
Node2vec-AE (proposal)		✓		✓	✓
Packet2vec-AE (proposal)	✓	✓	✓	✓	✓

6. Conclusions

In this paper, packet2vec learning algorithm is used to preprocess the network packet to obtain local proximity features that describe the similarity relationship between network packets. The local proximity features of the network packets are combined with the original features as the input of the deep auto-encoder for intrusion detection. The experiment proves that our proposed algorithms achieve higher accuracy than three of the state-of-the-art algorithms. In addition, it can be concluded from the experiment that the empirical formula, i.e., the pruning threshold $\varepsilon \approx$ the mean of weights in $G' - depth' *$ penalty value η, can be used to calculate the approximately optimal penalty value η. The optimal penalty value η is used to constrain the random walk range to extract features that accurately describe the local proximity of the network packet. Finally, these features are used to achieve optimal performance of IDS. In the best case, the accuracy, the detection rate, the precision, and the F_1 of packet2vec-AE are up to 94.7%, 90.9%, 94.3%, and 92.6%, respectively. In the worst case, the accuracy, the detection rate, the precision, and the F_1 of packet2vec-AE reached 87.4%, 81.1%, 87.3% and 84.1%, respectively. In the worst case, the proposed algorithm still achieves higher accuracy than three of the state-of-the-art algorithms.

Author Contributions: Conceptualization, Y.H. and Y.S.; Methodology, Y.H. and Y.S.; Software, Y.H., Y.S. and J.W.; Visualization, Y.H., Y.S. and J.W.; Writing—original draft, Y.H.; Writing—review & editing, Y.S. and J.W.

Funding: This work was funded by the Strategic Priority Research Program of Chinese Academy of Sciences under Grant XDC02020400.

Acknowledgments: We would like to thank Chaopeng Li, Yuan Wang, Jinlong Hao, Haijie Yin, Yi Liao, Teng Zeng, the editor and anonymous reviewers for their insightful comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Yin, C.; Zhu, Y.; Fei, J. A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks. *IEEE Access* **2017**, *5*, 21954–21961. [CrossRef]
2. Maimo, L.F.; Gomez, A.L.P.; Clemente, F.J.; Perez, M.G.; Perez, G.M. A Self-Adaptive Deep Learning-Based System for Anomaly Detection in 5G Networks. *IEEE Access* **2018**, *6*, 7700–7712. [CrossRef]
3. Panda, M.; Patra, M.R. A Comparative Study of Data Mining Algorithms for Network Intrusion Detection. In Proceedings of the 2008 First International Conference on Emerging Trends in Engineering and Technology, Nagpur, Maharashtra, India, 16–18 July 2008; pp. 81–83.
4. Goyal, P.; Ferrara, E. Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowl.-Based Syst.* **2017**, *151*, 78–94. [CrossRef]
5. Zhang, D.; Yin, J. Network Representation Learning: A Survey. *IEEE Trans. Big Data* **2017**. [CrossRef]
6. Hao, Y.; Sheng, Y.; Wang, J. Variant Gated Recurrent Units With Encoders to Preprocess Packets for Payload-Aware Intrusion Detection. *IEEE Access* **2019**, *7*, 49985–49998. [CrossRef]
7. Gouveia, A.; Correia, M. A Systematic Approach for the Application of Restricted Boltzmann Machines in Network Intrusion Detection. In *International Work-Conference on Artificial Neural Networks*; Springer: Cham, Switzerland, 2017.
8. Liu , G.; Yi, Z.; Yang, S. A hierarchical intrusion detection model based on the PCA neural networks. *Neurocomputing* **2007**, *70*, 1561–1568. [CrossRef]
9. Farahnakian, F.; Heikkonen, J. A deep auto-encoder based approach for intrusion detection system. In Proceedings of the 2018 20th International Conference on Advanced Communication Technology (ICACT), Chuncheon-si Gangwon-do, Korea, 11–14 February 2018.
10. Grover, A.; Leskovec, J. node2vec: Scalable Feature Learning for Networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; Volume 2016, pp. 855–864.
11. Harris, Z.S. *Distributional Structure*; Springer: Berlin, Germany, 1981.
12. Mikolov, T.; Chen, K.; Corrado, G. Efficient Estimation of Word Representations in Vector Space. *arXiv* **2013**, arXiv:1301.3781.
13. Keerthi, V.K.; Surendiran, B. Dimensionality reduction using principal component analysis for network intrusion detection. *Perspect. Sci.* **2016**, *8*, 510–512. [CrossRef]
14. Gulshan, K.; Kumar, K. Design of an Evolutionary Approach for Intrusion Detection. *Sci. World J.* **2013**, *5*, 962185.
15. Faraji, D.F.; Abbaspour, M. Extracting fuzzy attack patterns using an online fuzzy adaptive alert correlation framework. *Secur. Commun. Netw.* **2016**, *9*, 2245–2260. [CrossRef]
16. Abolhasanzadeh, B. Nonlinear dimensionality reduction for intrusion detection using auto-encoder bottleneck features. In Proceedings of the 2015 7th Conference on Information and Knowledge Technology (IKT), Urmia, Iran, 26–28 May 2015.
17. Singh, S.; Kaur, G. Unsupervised Anomaly Detection In Network Intrusion Detection Using Clusters. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science—Volume 38*; Australian Computer Society, Inc.: Newcastle, Australia, 2008.
18. Fan, W.; Bouguila, N.; Ziou, D. Unsupervised Anomaly Intrusion Detection via Localized Bayesian Feature Selection. In Proceedings of the 2011 IEEE 11th International Conference on Data Mining, Vancouver, BC, Canada, 11–14 December 2011.
19. Liao, Y.; Wang, Y.; Liu, Y. Graph Regularized Auto-Encoders for Image Representation. *IEEE Trans Image Process.* **2017**, *26*, 2839–2852. [CrossRef] [PubMed]

20. Perozzi, B.; Al-Rfou, R.; Skiena, S. DeepWalk: Online Learning of Social Representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 24–27 August 2014.
21. Tang, J.; Qu, M.; Wang, M. LINE: Large-scale Information Network Embedding. In Proceedings of the 24th International Conference on World Wide Web, Florence, Italy, 18–22 May 2015.
22. Hamilton, W.L.; Ying, R.; Leskovec, J. Representation Learning on Graphs: Methods and Applications. *arXiv* **2017**, arXiv:1709.05584.
23. Hao, Y.; Sheng, Y.; Wang, J.; Li, C. Network security event prediction based on recurrent neural network. *J. Netw. New Media* **2017**, *6*, 58–62. (In Chinese)
24. Garcia, V.; Bruna, J. Few-Shot Learning with Graph Neural Networks. *arXiv* **2017**, arXiv:1711.04043.
25. Perozzi, B.; Kulkarni, V.; Chen, H.; Skiena, S. Don't Walk, Skip! Online Learning of Multi-scale Network Embeddings. In Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017, Sydney, Australia, 31 July 31–3 August 2017.
26. Bhagat, S.; Cormode, G.; Muthukrishnan, S. Node Classification in Social Networks. *Comput. Sci.* **2011**, *16*, 115–148.
27. Wang, W.; Sheng, Y.; Wang, J.; Zeng, X.; Ye, X.; Huang, Y.; Zhu, M. Hast-ids: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. *IEEE Access* **2017**, *6*, 1792–1806. [CrossRef]
28. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed representations of words and phrases and their compositionality. *Adv. Neural Inf. Process. Syst.* **2013**, *26*, 3111–3119.
29. Tavallaee, M.; Bagheri, E.; Lu, W.; Ghorbani, A.A. A detailed analysis of the kdd cup 99 data set. In Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, Ottawa, ON, Canada, 8–10 July 2009.
30. Shanbhogue, R.D.; Beena, B.M. Survey of data mining (dm) and machine learning (ml) methods on cyber security. *Indian J. Sci. Technol.* **2017**, *10*, 1–7. [CrossRef]
31. Xu, Q.; Li, Z. The effect of different hidden unit number of sparse autoencoder. In Proceedings of the 27th Chinese Control and Decision Conference (2015 CCDC), Qingdao, China, 23–25 May 2015.
32. Lee, J.H.; Lee, J.H.; Sohn, S.G.; Ryu, J.H.; Chung, T.M. Effective Value of Decision Tree with KDD 99 Intrusion Detection Datasets for Intrusion Detection System. In Proceedings of the 2008 10th International Conference on Advanced Communication Technology, Gangwon-Do, Korea, 17–20 February 2008.
33. Tian, B.; Merrick, K.; Yu, S.; Hu, J. A hierarchical pea-based anomaly detection model. In Proceedings of the 2013 International Conference on Computing, Networking and Communications (ICNC), San Diego, CA, USA, 28–31 January 2013.
34. Kim, J.; Kim, J.; Thu, H.; Kim, H. Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection. In Proceedings of the 2016 International Conference on Platform Technology and Service (PlatCon), Jeju, Korea, 15–17 February 2016.
35. Zazo, R.; Nidadavolu, P.S.; Chen, N.; Rodriguez, J.G.; Dehak, N. Age estimation in short speech utterances based on lstm recurrent neural networks. *IEEE Access* **2018**, *6*, 22524–22530. [CrossRef]
36. Agarap, A.F. A neural network architecture combining gated recurrent unit (gru) and support vector machine (svm) for intrusion detection in network traffic data. In Proceedings of the 2018 10th International Conference on Machine Learning and Computing, Macau, China, 26–28 February 2018; pp. 26–30.
37. Imtiaz, U.; Qusay, H.M. A Filter-based Feature Selection Model for Anomaly-based Intrusion Detection Systems. In Proceedings of the 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 11–14 December 2017.
38. Baris, Y.; Guvensan, M.A.; Yavuz, A.G.; Karsligil, M.E. Application Identification via Network Traffic Classification. In Proceedings of the 2017 International Conference on Computing, Networking and Communications (ICNC), Santa Clara, CA, USA, 26–29 January 2017.
39. Gil, G.D.; Lashkari, A.H.; Mamun, M.; Ghorbani, A.A.; Chung, T.M. Characterization of encrypted and VPN traffic using time-related features. In Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016), Rome, Italy, 19–21 February 2016.
40. Coull, J.T.; Nazarian, B.; Vidal, F. Timing, Storage, and Comparison of Stimulus Duration Engage Discrete Anatomical Components of a Perceptual Timing Network. *J. Cogn. Neurosci.* **2008**, *20*, 2185–2197. [CrossRef]

41. Bjerregaard, T.; Stensgaard, M.B.; Sparsø, J. A scalable, timing-safe, network-on-chip architecture with an integrated clock distribution method. In Proceedings of the 2007 Design, Automation & Test in Europe Conference & Exhibition, Nice, France, 16–20 April 2007.
42. Du, Q.; Gong, G.; Pan, W. A packet-based precise timing and synchronous DAQ network for the LHAASO project. *Nuclear Instrum. Methods Phys. Res.* **2013**, *732*, 488–492. [CrossRef]
43. Matthew, H.; BooJoong, K.; Kieran, M.; Sakir, S. Peer Based Tracking using Multi-Tuple Indexing for Network Traffic Analysis and Malware Detection. In Proceedings of the 2018 16th Annual Conference on Privacy, Security and Trust (PST), Belfast, UK, 28–30 August 2018.
44. Zargar, G.R.; Kabiri, P. Selection of Effective Network Parameters in Attacks for Intrusion Detection. In Proceedings of the Industrial Conference on Advances in Data Mining: Applications & Theoretical Aspects, Berlin, Germany, 12–14 July 2010.
45. Wen, L.I.; Dai, Y.X.; Lian, Y.F.; Feng, P.M. Context Sensitive Host-Based IDS Using Hybrid Automaton: Context Sensitive Host-Based IDS Using Hybrid Automaton. *J. Softw.* **2009**, *20*,138–151.
46. Forootaninia, A.; Ghaznavighoushchi, M.B. An Improved Watchdog Technique Based On Power-Aware Hierarchical Design For Ids In Wireless Sensor Networks. *Int. J. Netw. Secur. Appl.* **2012**, *4*, doi:10.5121/ijnsa.2012.4411. [CrossRef]
47. Kachhvah, A.D.; Gupte, N. Transmission of packets on a hierarchical network: Statistics and explosive percolation. *Phys. Rev. E Stat. Nonlinear Soft Matter Phys.* **2012**, *86*, 026104. [CrossRef]
48. Chadli, S.; Emharraf, M.; Saber, M.; Ziyyat, A. Combination of Hierarchical and Cooperative Models of an IDS for MANETs. In Proceedings of the 2014 Tenth International Conference on Signal-Image Technology and Internet-Based Systems, Marrakech, Morocco, 23–27 November 2015.
49. Sheng, G.; Liu, J.W.; Hui, L.I.; Zhang, Q. A hierarchical IDS model for MANET based on weighted clustering with self-recommendation. *J. Xidian Univ.* **2007**, *34*, 279–284.
50. Subaira, A.S.; Anitha, P. Efficient classification mechanism for network intrusion detection system based on data mining techniques: A survey. In Proceedings of the 2014 IEEE 8th International Conference on Intelligent Systems and Control (ISCO), Coimbatore, India, 10–11 January 2015; pp. 274–280.
51. Sallay, H.; Ammar, A.; Saad, M.B.; Bourouis, S. A real time adaptive intrusion detection alert classifier for high speed networks. In Proceedings of the 2013 IEEE 12th International Symposium on Network Computing and Applications, Cambridge, MA, USA, 22–24 August 2013; pp. 73–80.
52. Noorbehbahani, F.; Fanian, A.; Mousavi, R.; Hasannejad, H. An incremental intrusion detection system using a new semi-supervised stream classification method. *Int. J. Commun. Syst.* **2017**, *30*, e3002. [CrossRef]
53. Villalba, L.J.; Castro, J.D.M.; Orozco, A.L.S.; Puentes, J.M. Malware Detection System by Payload Analysis of Network Traffic. In *International Workshop on Recent Advances in Intrusion Detection*; Springer: Berlin/Heidelberg, Germany, 2012.
54. Kaur, R.; Nagpal, E.S.; Chamotra, S. Malicious traffic detection in a private organizational network using honeynet system. In Proceedings of the 2015 Annual IEEE India Conference (INDICON), New Delhi, India, 17–20 December 2016.
55. Hindy, H.; Hodo, E.; Bayne, E.; Seeam, A.; Bellekens, X. A Taxonomy of Malicious Traffic for Intrusion Detection Systems. In Proceedings of the 2018 International Conference On Cyber Situational Awareness, Data Analytics And Assessment (Cyber SA), Glasgow, UK, 11–12 June 2018.