MINISTRY OF NATIONAL DEFENSE
MILITARY TECHNICAL ACADEMY

VU THI LY

DEVELOPING DEEP NEURAL NETWORKS FOR NETWORK ATTACK DETECTION

Major: Mathematical Foundation for Informatics
Code: 9 46 01 10

SUMMARY OF TECHNICAL DOCTORAL THESIS

HA NOI - 2021

THIS WORK IS COMPLETED AT MILITARY TECHNICAL ACADEMY - MINISTRY OF NATIONAL DEFENSE

Supervisor:

1. **Assoc. Prof. Dr. Nguyen Quang Uy**
2. **Prof. Dr. Eryk Duzkite**

Opponent 1:

Assoc. Prof. Dr. Nguyen Duc Dung

Opponent 2:

Assoc. Prof. Dr. Nguyen Thi Thuy

Opponent 3:

Assoc. Prof. Dr. Nguyen Hieu Minh

This thesis will be defended before The Academy-Level Doctoral Examination Board according to the Decision No 1814/QD-HV date 18 month 5 year 2021 of the President of Military Technical Academy, meeting at the Military Technical Academy at time ... date ... month ... year .....

This thesis could be found at:

- National Library of Vietnam
- Library of Military Technical Academy

PUBLICATIONS

[Paper 1] **Ly Vu**, Cong Thanh Bui, and Nguyen Quang Uy: *A deep learning based method for handling imbalanced problem in network traffic classification.* In: Proceedings of the Eighth International Symposium on Information and Communication Technology. pp. 333–339. ACM (Dec. 2017).

[Paper 2] **Ly Vu**, Van Loi Cao, Quang Uy Nguyen, Diep N. Nguyen, Dinh Thai Hoang, and Eryk Dutkiewicz: *Learning Latent Distribution for Distinguishing Network Traffic in Intrusion Detection System.* IEEE International Conference on Communications (ICC), Rank B, pp. 1–6 (2019).

[Paper 3] **Ly Vu** and Quang Uy Nguyen: *An Ensemble of Activation Functions in AutoEncoder Applied to IoT Anomaly Detection.* In: The 2019 6th NAFOSTED Conference on Information and Computer Science (NICS'19), pp. 534–539 (2019).

[Paper 4] **Ly Vu**, Quang Uy Nguyen, Diep N. Nguyen, Dinh Thai Hoang, and Eryk Dutkiewicz:*Deep Transfer Learning for IoT Attack Detection.* In: IEEE Access (ISI-SCIE, IF = 3.745). pp.1-10, June 2020.

[Paper 5] **Ly Vu** and Quang Uy Nguyen: *Handling Imbalanced Data in Intrusion Detection Systems using Generative Adversarial Networks.* In: Journal of Research and Development on Information and Communication Technology. Vol. 2020, no. 1, Sept. 2020.

[Paper 6] **Ly Vu**, Van Loi Cao, Quang Uy Nguyen, Diep N. Nguyen, Dinh Thai Hoang, and Eryk Dutkiewicz: *Learning Latent Representation for IoT Anomaly Detection.* In: IEEE Transactions on Cybernetics (ISI-SCI, IF=11.079), DOI: 10.1109/TCYB.2020.3013416, Sept. 2020.

CONCLUSIONS AND FUTURE WORK

Contributions

In addition to a review of literature regarding to the research in this thesis, the following main contributions can be drawn from the investigations presented in the thesis:

1. First, to effectively detect new/unknown attacks by machine learning methods, we propose a novel representation learning method to better predictively "describe" unknown attacks, facilitating the subsequent machine learning-based network attack detection.
2. Second, we handle the imbalance problem of network attack datasets. We propose a novel solution to this problem by using generative adversarial networks to generate synthesized attack data for network attack data.
3. Third, we resolve "the lack of label information" in the NAD problem. We develop a TL technique that can transfer the knowledge of label information from a domain (i.e., data collected from one IoT device) to a related domain (i.e., data collected from a different IoT device) without label information.

Limitations

1. Training time of proposed models is time consuming.
2. In CDAAE, we need to assume that the original data distribution follows a Gaussian distribution.
3. Training MMD-AE is more time consuming than previous TL models due to transferring processes executed in multiple layers.

Future work

1. In the CDAAE model, we can explore other distributions different from the Gaussian distribution that may better represent the original data distribution. Moreover, we expect that by adding some attributes of malicious behaviors to CDAAE.
2. We will distribute the training process of the proposed DTL model to the multiple IoT nodes by the federated learning technique to speed up this process.

INTRODUCTION

1. Motivation:

The unprecedented development of communication networks has significant contributions for human beings but also places many challenges for information security problems due to the diversity of emerging cyberattacks. As a result, early detecting network attacks plays a crucial role in preventing cyberattacks. A network attack detection (NAD) monitors the network traffic to identify abnormal activities in the network environments.

Machine learning, especially deep learning, has achieved remarkable success in network attack detection, there are still some unsolved problems. First, the network traffic is heterogeneous due to the diversity of network environments. Second, network attack datasets are usually highly imbalanced. Third, in some network environments, e.g., IoT, the data distribution in one device may be very different from that in other devices. Consequently, this thesis focuses on three main objectives which can enhance NAD based on supervised machine learning methods.

2. Thesis contributions:

1. The thesis proposes three latent representation learning models based on AutoEncoders (AEs) to project normal traffic data and attack traffic data.
2. The thesis proposes three new deep generative models for handling data imbalance, thereby improving the accuracy of machine learning methods for NAD systems.
3. A DTL model is proposed to handle the "lack of label information" problem in training data.

3. Thesis Structure:

The thesis includes four main content chapters. Chapter 1 presents the fundamental background of the NAD problem and deep neural techniques. Chapter 2 proposes a new latent representation learning technique that helps network attacks to be detected more easily. Chapter 3 presents new generative deep neural network models for handling the imbalance of network traffic datasets. Chapter 4 proposes a new DTL model based on a deep neural network.

Chapter 1

BACKGROUNDS

The Internet becomes an essential function in our living. Simultaneously, while the Internet does us excellent service, it also raises many security threats. Security attacks have become a crucial portion that restricts the growth of the Internet. Recently, NAD methods have received considerable attention recently to guarantee the security of information systems.

Security data indicate the network traffic data that can be used to detect security attacks. Network attack detection methods are highly based on security data. The quality of security data affects the performance of NAD methods. The network security datasets usually represent network traffic using both packet-based features and flow-based features. Under the scope of this thesis, we focus on the methods to identify the network attacks in published datasets instead of analysing data generated from network attacks.

Deep neural networks provide a robust framework for learning data features. A deep neural network aims to map an input vector to an output vector where the output vector is easier for people as well as other machine learning tasks. This mapping is done by given large models and large labeled training data samples. In this thesis, the proposed deep neural network models based on two main baseline models, i.e., AutoEncoder (AE) and Adversarial AutoEncoder (AAE).

This thesis also proposes a Deep Transfer Learning (DTL) model based on AE for NAD. DTL is used to transfer knowledge learned from a source domain to a target domain where the target domain is different with the source domain but they are related data distributions. In this thesis, the proposed DTL can even handle the problems of having less data or no label information in the target domain.

Table 4.2: Processing time and complexity of DTL models.

Models	**Training Time (hours)**	**Predicting Time (second)**	**No. Parameters**
AE	0.001	1.001	25117
SKL-AE	0.443	1.112	150702
SMD-AE	3.693	1.110	150702
MMD-AE	11.057	1.108	150702

4.3.2 Performance Comparison

Table 4.1 represents the AUC scores of AE and three DTL models based on AE. The first DTL model is the DTL model that transfers knowledge on only single layer using Kullback Libler divergence metric (SKL-AE). The second DTL model is the DTL model that transfers knowledge on only single layer using MMD metric (SMD-AE). Third, the proposed DTL model, i.e., MMD-AE. All these models are trained on the dataset with label information in the columns and the dataset without information in the rows and tested on the dataset in the rows.

We can observe that our proposed DTL model usually achieves the highest AUC score in almost all IoT datasets[1].This result proves that implementing the transferring task in multiple layers of MMD-AE helps the model transfers more effectively the label information from the source to the target domain.

4.3.3 Processing Time and Complexity Analysis

Table 4.2 shows the training and the predicting time of the tested model when the source domain is IoT-2, and the target domain is IoT-1. It can be seen that the training process of the DTL methods is more time consuming than that of AE.Moreover, the training processes present the same number of trainable parameters for all the DTL models based on AE. However, more important is that the predicting time of all DTL methods is mostly equal to that of AE.

4.4 Conclusion

In this chapter, we have introduced a novel DTL-based approach for IoT network attack detection, namely MMD-AE. This proposed approach aims to address the problem of "lack of labeled information" for the training detection model in ubiquitous IoT devices.

[1]The AUCs of the proposed model in each scenario is presented by the bold text style.

Table 4.1: AUC score comparision of AE, SKL-AE, SMD-AE and MMD-AE.

Target	Model	Source								
		IoT-1	**IoT-2**	**IoT-3**	**IoT-4**	**IoT-5**	**IoT-6**	**IoT-7**	**IoT-8**	**IoT-9**
IoT-1	AE		0.705	0.542	0.768	0.838	0.643	0.791	0.632	0.600
	SKL-AE		0.700	0.759	0.855	0.943	0.729	0.733	0.689	0.705
	SMD-AE		0.722	0.777	0.875	0.943	0.766	0.791	0.701	0.705
	MMD-AE		**0.888**	**0.796**	**0.885**	**0.943**	**0.833**	**0.892**	**0.775**	**0.743**
IoT-2	AE	0.540		0.500	0.647	0.509	0.743	0.981	0.777	0.578
	SKL-AE	0.545		0.990	0.708	0.685	0.794	0.827	0.648	0.606
	SMD-AE	0.563		0.990	0.815	0.689	0.874	0.871	0.778	0.607
	MMD-AE	**0.937**		**0.990**	**0.898**	**0.692**	**0.878**	**0.900**	**0.787**	**0.609**
IoT-3	AE	0.600	0.659		0.530	0.500	0.501	0.644	0.805	0.899
	SKL-AE	0.745	0.922		0.566	0.939	0.534	0.640	0.933	0.916
	SMD-AE	0.764	0.849		0.625	0.879	0.561	0.600	0.918	0.938
	MMD-AE	**0.937**	**0.956**		**0.978**	**0.928**	**0.610**	**0.654**	**0.937**	**0.946**
IoT-4	AE	0.709	0.740	0.817		0.809	0.502	0.944	0.806	0.800
	SKL-AE	0.760	0.852	0.837		0.806	0.824	0.949	0.836	0.809
	SMD-AE	0.777	0.811	0.840		0.803	0.952	0.947	0.809	0.826
	MMD-AE	**0.937**	**0.857**	**0.935**		**0.844**	**0.957**	**0.959**	**0.875**	**0.850**
IoT-5	AE	0.615	0.598	0.824	0.670		0.920	0.803	0.790	0.698
	SKL-AE	0.645	0.639	0.948	0.633		0.923	0.695	0.802	0.635
	SMD-AE	0.661	0.576	0.954	0.672		0.945	0.822	0.789	0.833
	MMD-AE	**0.665**	**0.508**	**0.954**	**0.679**		**0.928**	**0.847**	**0.816**	**0.928**
IoT-6	AE	0.824	0.823	0.699	0.834	0.936		0.765	0.836	0.737
	SKL-AE	0.861	0.897	0.711	0.739	0.980		0.893	0.787	0.881
	SMD-AE	0.879	0.898	0.713	0.849	0.982		0.778	0.867	0.898
	MMD-AE	**0.927**	**0.899**	**0.787**	**0.846**	**0.992**		**0.974**	**0.871**	**0.898**
IoT-7	AE	0.504	0.501	0.626	0.791	0.616	0.809		0.598	0.459
	SKL-AE	0.508	0.625	0.865	0.831	0.550	0.906		0.358	0.524
	SMD-AE	0.519	0.619	0.865	0.817	0.643	0.884		0.613	0.604
	MMD-AE	**0.548**	**0.621**	**0.888**	**0.897**	**0.858**	**0.905**		**0.615**	**0.618**
IoT-8	AE	0.814	0.599	0.831	0.650	0.628	0.890	0.901		0.588
	SKL-AE	0.619	0.636	0.892	0.600	0.629	0.923	0.907		0.712
	SMD-AE	0.622	0.639	0.902	0.717	0.632	0.919	0.872		0.629
	MMD-AE	**0.735**	**0.636**	**0.964**	**0.723**	**0.692**	**0.977**	**0.943**		**0.616**
IoT-9	AE	0.823	0.601	0.840	0.851	0.691	0.808	0.885	0.579	
	SKL-AE	0.810	0.602	0.800	0.731	0.662	**0.940**	0.855	0.562	
	SMD-AE	0.830	0.609	0.892	0.600	0.901	0.806	0.886	0.626	
	MMD-AE	**0.843**	**0.911**	**0.910**	**0.874**	**0.904**	0.829	**0.889**	**0.643**	

Chapter 2

LEARNING LATENT REPRESENTATION FOR NETWORK ATTACK DETECTION

2.1 Introduction

In this chapter, we propose a novel representation learning method to enhance the accuracy of deep learning in detecting network attacks, especially unknown attacks. Specifically, we develop the regularized versions of AutoEncoders (AEs) to learn a new representation space for the input data. In the new representation space, normal data and known network attacks will be forced into two tightly separated regions, called normal region and anomalous region. We hypothesize that unknown attacks will appear closer to the anomalous region as they may share some common characteristics with known ones. Hence, they can be easily detected.

2.2 Proposed Representation Learning Model

2.2.1 Muti-distribution Variational AutoEncoder

Muti-distribution Variational AutoEncoder (MVAE) is a regularized version of Variational AutoEncoder (VAE), aiming to learn the probability distributions representing the input data. To that end, this chapter incorporates the label information into the loss function of VAE to represent data into two Gaussian distributions with different mean values. Given a data sample x^i with its associated label y^i, μ_{y^i} is the distribution centroid for the class y^i. The loss function of MVAE on x^i can be calculated as follows:

$$\begin{aligned}\ell_{MVAE}(x^i, y^i, \theta, \phi) = -\frac{1}{K}\sum_{k=1}^{K} \log p_\theta(x^i|z^{i,k}, y^i) \\ + D_{KL}(q_\phi(z^i|x^i, y^i)||p(z^i|y^i)),\end{aligned} \tag{2.1}$$

where $z^{i,k}$ is reparameterized as $z^{i,k} = \mu_{y^i} + \sigma^i\epsilon^k$ and $\epsilon^k \sim \mathcal{N}(0, 1)$; K and y^i are the number of samples used to reparameterize x^i and the label of the sample x^i, respectively.

The loss function of MVAE consists of two terms. The first term is Reconstruction Error (RE) or the expected negative log-likelihood of the i-th data point to reconstruct the original data at its output layer. The second term

is created by incorporating the label information to the posterior distribution $q_\phi(z^i|x^i)$ and the prior distribution $p(z^i)$ of VAE. Therefore, the second term is the Kullback Leibler (KL) divergence between the approximate distribution $q_\phi(z^i|x^i, y^i)$ and the conditional distribution $p(z^i|y^i)$. The objective of adding the label information to the second term is to force the samples from each class data to reside in each Gaussian distribution conditioned on the label y^i. Moreover, $p(z^i|y^i)$ follows the normal distribution with the mean μ_{y^i} and the standard deviation 1.0, $p(z^i|y^i) = \mathcal{N}(\mu_{y^i}, 1)$. The posterior distribution $q_\phi(z^i|x^i, y^i)$ is the multi-variate Gaussian with a diagonal covariance structure. In other words, $q_\phi(z^i|x^i, y^i) = \mathcal{N}(\mu^i, (\sigma^i)^2)$, where μ^i and σ^i are the mean and standard deviation, respectively, are sampled from the sample x^i. Thus, the Multi-KL term in Eq. 2.1 is rewritten as follows:

$$\begin{aligned} &D_{KL}(q_\phi(z^i|x^i, y^i)||p_\theta(z^i|y^i)) \\ &\qquad = D_{KL}(\mathcal{N}(\mu^i, (\sigma^i)^2)||\mathcal{N}(\mu_{y^i}, 1)). \end{aligned} \tag{2.2}$$

Let D, μ^i_j and σ^i_j denote the dimension of z^i, the j-th element of μ^i and σ^i, respectively; $\mu_{y^i_j}$ is the j-th element of μ_{y^i}. Then, applying the computation of the KL divergence, the Multi-KL term is rewritten as follows:

$$\begin{aligned} &D_{KL}\left(q_\phi(z|x^i, y^i)\|p_\theta(z|y^i)\right) \\ &\qquad = \frac{1}{2}\sum_{j=1}^{D}\Big((\sigma^i_j)^2 + (\mu^i_j - \mu_{y^i_j})^2 - 1 - \log((\sigma^i_j)^2)\Big). \end{aligned} \tag{2.3}$$

Taking Multi-KL term in Eq. 2.3, the loss function of MVAE in Eq. 2.1 finally is rewritten as follows:

$$\ell_{MVAE}(x^i, y^i, \theta, \phi) = -\frac{1}{K}\sum_{k=1}^{K}\log p_\theta(x^i|z^{i,k}, y^i) + \lambda\frac{1}{2}\sum_{j=1}^{D}((\sigma^i_j)^2 + (\mu^i_j - \mu_{y^i_j})^2 - 1 - \log((\sigma^i_j)^2)), \tag{2.4}$$

where λ is a parameter to control the trade-off between two terms in Eq. 2.4.

The mean values for the distributions of the normal class and attack class are chosen to make these distributions located far enough from each other. In our experiments, the mean values are 4 and 12 for the normal class and attack class, respectively. These values are calibrated from the experiments for the good performance of MVAE. In this chapter, the distribution centroid μ_{y^i} for the class y^i, and the trade-off parameter λ are determined in advance. The

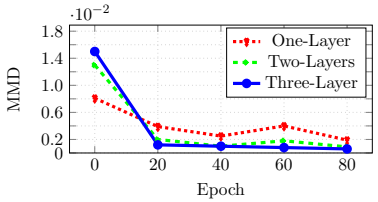


Fig. 4.2: MMD of transferring task (from IoT-1 to IoT-2) on one, two, and three encoding layers.

and target data close together. The ℓ_{MMD} loss term is described as follows:

$$\ell_{\text{MMD}}(x^i_S, \phi_S, \theta_S, x^i_T, \phi_T, \theta_T) = \sum_{k=1}^{K}\text{MMD}(\xi^k_S(x^i_S), \xi^k_T(x^i_T)), \tag{4.3}$$

where K is the number of encoding layers in the AE-based model. $\xi^k_S(x^i_S)$ and $\xi^k_T(x^i_T)$ are the encoding layers k-th of AE_1 and AE_2, respectively, $\text{MMD}(,)$ is the MMD distance between two data distributions.

The final loss function of MMD-AE combines the loss terms in Eq. 4.1, Eq. 4.2, and Eq. 4.3 as in Eq. 4.4.

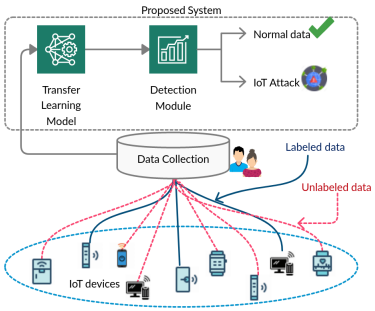
$$\ell = \ell_{\text{SE}} + \ell_{\text{RE}} + \ell_{\text{MMD}}. \tag{4.4}$$

Compared with previous TL models, MMD-AE can transfer the knowledge not only in the bottleneck layer but also in *every encoding layer* from the source domain AE_1, to the target domain, AE_2.

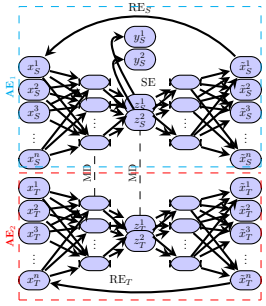
4.3 Results and Discussions

4.3.1 Effectiveness of Transferring Information in MMD-AE

We measured the MMD distance between the latent representation, i.e., the bottleneck layer, of AE_1 and AE_2 when the transfer information is implemented in one, two and three layers of the encoders. The smaller distance, the more information is transferred from the source domain (AE_1) to the target domain (AE_2). The result is presented in Fig. 4.2. This result evidences that our proposed solution MMD-AE is more effective than the previous DTL models that perform the transferring task only on the bottleneck layer of AE.



(a) Proposed system structure.



(b) Architecture of MMD-AE.

Fig. 4.1: Deep Transfer Learning Model.

the data samples of input layers and the output layers of the source domain and the target domain, respectively.

The second term ℓ_{SE} aims to train a classifier at the latent representation of AE_1 using labeled information in the source domain. In other words, this term attempts to map the value at two neurons at the bottleneck layer of AE_1, i.e., z_S, to their label information y_S. This is achieved by using the softmax function to minimize the difference between z_S and y_S. This loss encourages to distinguish the latent representation space from separated class labels. Formally, this loss is defined as follows:

$$\ell_{\text{SE}}(x_S^i, y_S^i, \phi_S.\theta_S) = -\sum_{j=1}^{C} y_S^{i,j} \log(z_S^{i,j}), \tag{4.2}$$

where z_S^i and y_S^i are the latent representation and labels of the source data sample x_S^i. $y_S^{i,j}$ and $z_S^{i,j}$ represent the $j-th$ element of the vector y_S^i and z_S^i, respectively.

The third term ℓ_{MMD} is to transfer the knowledge of the source domain to the target domain. The transferring process is executed by minimizing the MMD distances between every encoding layers of AE_1 and the corresponding encoding layers of AE_2. This term aims to make the representations of the source data

hyper-parameter μ_{y^i} can receive two values associated with the normal class and the attack class.

2.2.2 *Multi-distribution AutoEncoder*

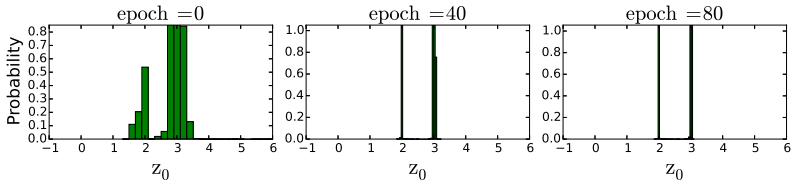


Fig. 2.1: The probability distribution of the latent data ($\mathbf{z}_0$) of MAE in the training process.

This subsection describes how to integrate a regularizer to an AE to create Multi-distribution AutoEncoder (MAE). The regularizer is a multi-distribution penalty, called $\Omega(\mathbf{z})$, on the latent representation $\mathbf{z}$. The penalty $\Omega(\mathbf{z})$ encourages the MAE to construct a new latent feature space in which each class of data is projected into a small region. Specifically, this chapter incorporates class labels into $\Omega(\mathbf{z})$ to restrict the data samples of each class to lie closely together centered at a pre-determined value. The new regularizer is presented in Eq. 2.5.

$$\Omega(\mathbf{z}) = \left|\left|\mathbf{z} - \mu_{y^i}\right|\right|^2, \tag{2.5}$$

where $\mathbf{z}$ is the latent data at the bottleneck layer of MAE, and μ_{y^i} is a distribution centroid of class y^i in the latent space. The label y^i used in $\Omega(\mathbf{z})$ maps the input data into its corresponding region defined by μ_{y^i} in the latent representation. The latent feature space is represented by multiple distributions based on the number of classes. Thus, this chapter names the new regularized AE to be Multi-distribution AE.

In the MAE loss function, this chapter also uses a parameter λ to control the trade-off between the RE and $\Omega(\mathbf{z})$ terms as discussed in Sub-section 2.2.1. Thus, the loss function of MAE can be defined as follows:

$$\ell_{MAE}(\theta, \phi, \mathbf{x}, \mathbf{z}) = \frac{1}{n}\sum_{i=1}^{n}\left(x^i - \hat{x}^i\right)^2 + \lambda\frac{1}{n}\sum_{i=1}^{n}\left|\left|z^i - \mu_{y^i}\right|\right|^2, \tag{2.6}$$

where x^i, z^i and $\hat{x}^i$ are the i-th element of the input samples, its corresponding latent data and reconstruction data, respectively. y^i and μ_{y^i} are the label of the sample x^i and the centroid of class y^i, respectively. n is the number of training samples. The first term in Eq. 2.6 is the RE that measures the difference between the input data and its reconstruction. The second term is the regularizer used to compress the input data to the separated regions in the latent space.

To visualize the probability distribution of the latent representation of MAE, i.e., $\mathbf{z}$, we calculate the histogram of one feature of the latent data $\mathbf{z_0}$. Fig. 2.1 presents the probability distribution of $\mathbf{z_0}$ of normal class and known attacks during the training process of MAE on the IoT-1 dataset. After some epochs, the latent data is constrained into two tight regions in the latent representation of MAE.

2.2.3 Multi-distribution Denoising AutoEncoder

In this subsection, this chapter discusses the details of the multi-distribution Denoising AutoEncoder (MDAE). In this chapter, this chapter employs DAE to develop MDAE. For each data sample x^i, we can draw its corrupted version $\tilde{x}^i$ by a Gaussian noise. MDAE learns to reconstruct the original input x^i from a corrupted data $\tilde{x}^i$, and also penalizes the corresponding latent vector z^i to be close to μ_{y^i}. The loss function of MDAE can be presented in Eq. 2.7.

$$\ell_{MDAE}(\mathbf{x}, \tilde{\mathbf{x}}, \mathbf{z}, \phi, \theta) = \frac{1}{n}\sum_{i=1}^{n}\left(x^i - p_\theta(q_\phi(\tilde{x}^i))\right)^2 + \lambda\frac{1}{n}\sum_{i=1}^{n}\left||z^i - \mu_{y^i}\right||^2, \tag{2.7}$$

where z^i is the latent vector of the data sample x^i. μ_{y^i} is the predefined distribution centroid of the class y^i in the latent feature space of MDAE. q_ϕ and p_θ are the encoder and decoder parts as in DAE, respectively. n is the number of training samples. The hyper-parameter λ controls the trade-off between two terms in Eq. 2.7.

2.3 Results and Analysis

2.3.1 Ability to Detect Unknown Attacks

We evaluate the proposed models based on the ability to detect unknown attacks of the four classifiers trained on the latent representation. As mentioned above, each of the nine IoT datasets has five or ten specific types of botnet attacks. For each IoT dataset, we randomly select two types of IoT attacks, and 70% of normal traffic for training, and the rest of IoT attacks and normal data

Chapter 4

DEEP TRANSFER LEARNING FOR NETWORK ATTACK DETECTION

4.1 Introduction

The solutions proposed in previous chapters are based on the assumption that we can collect labeled data of both normal and attack classes. However, in some problem domains, it is often unable to label data for all samples collected from multiple devices. In this chapter, we propose a novel deep transfer learning (DTL) method to handle "lack of label information" in network attack datasets.

4.2 Proposed Deep Transfer Learning Model

Fig. 4.1(a) presents the system structure that uses DTL for IoT attack detection. First, the data collection module gathers data from all IoT devices. Second, the collected data is passed to the DTL model for training. After training, the trained DTL model is used in the detection module that can classify incoming traffic from all IoT devices.

The proposed DTL model named as Maximum Mean Discrepancy-AutoEncoder (MMD-AE) includes two AEs (i.e., AE_1 an AE_2) that have the same architecture as Fig. 4.1(b). The input of AE_1 is the data samples from the source domain (x_S^i), while the input of AE_2 is the data samples from the target domain (x_T^i). The training process attempts to minimize the MMD-AE loss function. This loss function includes three terms: the reconstruction error (ℓ_{RE}) term, the supervised (ℓ_{SE}) term and the Multi-Maximum Mean Discrepancy (ℓ_{MMD}) term.

We assume that ϕ_S, θ_S, ϕ_T, θ_T are the parameter sets of encoder and decoder of AE_1 and AE_2, respectively. The first term, ℓ_{RE} including RE_S and RE_T in Fig. 4.1(b) attempts to reconstruct the input layers at the output layers of both AEs. In other words, the RE_S and RE_T try to reconstruct the input data x_S and x_T at their output from the latent representations z_S and z_T, respectively. Formally, the ℓ_{RE} term is calculated as follows:

$$\ell_{\text{RE}}(x_S^i, \phi_S, \theta_S, x_T^i, \phi_T, \theta_T) = l(x_S^i, \hat{x}_S^i) + l(x_T^i, \hat{x}_T^i), \tag{4.1}$$

where l function is the mean squared error (MSE) function, x_S^i, $\hat{x}_S^i$, x_T^i, $\hat{x}_T^i$ are

Table 3.2: Log-likelihood estimation of generative models.

Model	NSL-KDD	UNSW-NB15	CTU13.6-Menti	CTU13.12-NSIS.ay	CTU13.13-Virut
SMOTE-SVM	15.87±0.69	22.84±0.67	64.07±0.96	52.80±1.36	52.15 ± 0.89
CVAE	65.65±0.33	40.59±1.03	95.68±1.47	92.78±0.29	104.83± 0.89
SAAE	68.20± 3.16	44.26±2.58	112.63±2.14	95.20±4.48	112.19±3.22
ACGAN	60.86±4.82	33.76±2.59	89.55±5.39	82.76±7.98	90.96±2.98
CDAAE	80.34±2.56	61.48±3.95	118.32±2.03	96.67±1.56	131.69 ± 1.84

generative models.

3.3.3 Complexity of Proposed Models

Table 3.3: Processing time of training and generating processes in seconds.

Methods	**NSL-KDD**			**UNSW-NB15**		
	Train Time	Generate Time	No. Parameters	Train Time	Generate Time	No. Parameters
SMOTE-SVM	109.3	0.2		98.9	0.2	
BalanceCascade	134.9	1.2		112.5	1.2	
CVAE	6345.6	0.3	174050	5154.4	1.4	837535
ACGAN	6402.5	0.4	115320	5024.8	2.2	823868
SAAE	5906.1	0.3	353967	5130.2	0.7	976058
ACGAN-SVM	9342.5	0.4	115320	8234.9	1.8	823868
CDAAE	6043.9	0.4	532946	4975.1	2.6	1132307
CDAAE-KNN	8576.7	0.9	532946	7456.8	3.7	1132307

We measured the computational time for training and generating synthesized samples. We can see that CDAAE-KNN often requires a longer time to generate data than the others. However, both training and generating processes are executed offline. Moreover, Table 3.3 proves that the proposed models increase the model size of the original ones due to the higher number of trainable parameters.

3.4 Conclusion

This chapter proposed three models to address the imbalance problem of network attack datasets. The CDAAE model is used to generate samples for a specific class label where ACGAN-SVM and CDAAE-KNN are used to synthesize samples that are close to the borderline between classifiers. The augmented datasets are used for other classification tasks.

are used for evaluating our models. We only use two types of DDoS attacks for training, and the rest is for testing. This guarantees that there are some types of IoT attacks used for evaluating models that have not been seen in the training process.

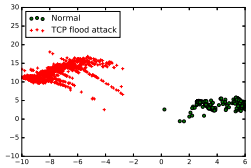
It can be seen from Table 2.1 that the latent representations resulting from MVAE, MAE, and MDAE help four classifiers achieve higher classification accuracy in comparison to those using the original data. For example, the AUC scores of Linear Support Vector Machine (SVM), Perceptron (PCT), Nearest Centroid (NCT), and Linear Regression (LR) working on the latent representation of MAE are increased from 0.839, 0.768, 0.743, and 0.862 to 0.999, 0.996, 0.998, and 0.999 with those working on the original data on the IoT-1 dataset, respectively. The increase in the classification accuracy can also be observed from MDAE and MVAE. Thus, four classifiers trained on the latent representations of MAE and MDAE tend to produce more consistent results than the previous one (MVAE).

We also carried out an experiment to explain why our models can support conventional classifiers to detect unknown attacks efficiently. Fig. 2.2 (a) and Fig. 2.2 (b) show that the representation of AE still can distinguish the normal samples, known attack samples and unknown attack samples. This is the main reason for the high performance of classifiers on the AE's representation presented in Table 2.1. However, while MAE can compress normal and known attack samples into two very compact areas on both the training and testing data, AE does not obtain this result. The normal and known attacks in the training data of AE spread significantly wider than the samples of MAE. More interestingly, the samples of unknown attacks in the testing data of MAE are mapped closely to the region of known attacks. Hence, they can be distinguished from normal samples easier.

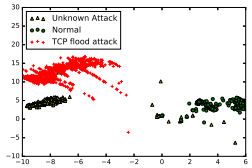
2.3.2 Cross-datasets Evaluation

This experiment aims to exam the stability of the latent representation produced by MVAE, MAE, and MDAE when training on one botnet family and evaluating the other. These guarantee that the testing attack family has not been seen in the training phase.

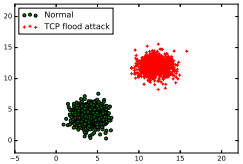
Table 2.2 shows that when the training data and testing data come from different botnet families, it is difficult for the NCT classifier to detect unknown botnet attacks. Both the standalone NCT (STA) and NCT with the representation of AE and DBN, tend to produce a poor performance in both scenarios.



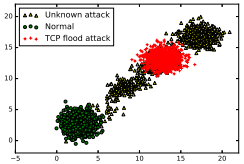
(a) Training samples of AE.



(b) Testing samples of AE.



(c) Training samples of MAE.



(d) Testing samples of MAE.

Fig. 2.2: Latent representation resulting from AE (a,b) and MAE (c,d).

On the other hand, the latent representations of MVAE, MAE, and MDAE are designed to reserve some regions being close to the anomaly region for unknown IoT attacks. These results confirm that our learning representation models can enhance the ability to detect unknown IoT attacks of simple classifiers.

2.3.3 Influence of the Hyper-parameters of Classifiers

This experiment investigates the influence of the hyper-parameters on the performance of classifiers when they are trained on the original feature space and the latent representation of five deep learning models including AE, DBN, MVAE, MAE and MDAE. The first parameter is analyzed as the hyper-parameter C of SVM and the second parameter is the distance *metric* used in the NCT classifier. Fig. 2.3 (a, b) shows that the SVM and NCT classifiers working with MVAE, MAE, and MDAE tends to yield high and stable AUC scores over the different values of the C and *metric* parameters, respectively.

The experiments in this subsection clearly show that our proposed models can support classifiers to perform consistently on a wide range of hyper-parameter settings. Thus, linear classifiers can easily distinguish attacks from normal data, and its performance is less sensitive to hyper-parameters.

Table 3.1: Results of SVM, DT, and RF of on the network attack datasets.

Alg.	Augmented datasets	NSL-KDD	UNSW-NB15	CTU13.6-Menti	CTU13.12-NSIS.ay	CTU13.13-Virut
SVM	ORIGINAL	0.570	0.129	0.500	0.500	0.831
	SMOTE-SVM	0.688	0.218	0.820	**0.824**	0.887
	BalanceCascade	0.620	0.304	0.876	0.820	0.895
	CVAE	0.736	0.325	0.927	0.630	0.950
	SAAE	0.738	0.345	0.928	0.653	0.951
	ACGAN	0.712	0.200	0.905	0.601	0.947
	ACGAN-SVM	0.752	0.205	0.932	0.658	0.953
	CDAAE	0.741	0.416	0.963	0.693	0.962
	CDAAE-KNN	**0.753**	**0.441**	**0.972**	0.702	**0.971**
DT	ORIGINAL	0.430	0.221	0.500	0.777	0.962
	SMOTE-SVM	0.446	0.348	0.892	0.782	0.964
	BalanceCascade	0.522	0.486	0.897	0.786	0.965
	CVAE	0.612	0.538	0.992	0.796	0.968
	SAAE	0.629	0.542	0.920	0.800	0.968
	ACGAN	0.523	0.506	0.905	0.799	0.967
	ACGAN-SVM	0.601	0.584	0.909	**0.834**	0.958
	CDAAE	0.650	0.592	0.934	0.816	0.971
	CDAAE-KNN	**0.660**	**0.598**	**0.941**	0.823	**0.976**
RF	ORIGINAL	0.760	0.357	0.500	0.846	0.951
	SMOTE-SVM	0.780	0.436	0.945	0.882	0.954
	BalanceCascade	0.793	0.439	0.949	0.888	0.956
	CVAE	0.824	0.572	0.958	0.921	0.958
	SAAE	0.823	0.571	0.956	0.922	0.956
	ACGAN	0.804	0.448	0.954	0.892	0.958
	ACGAN-SVM	0.834	0.589	0.962	0.901	0.965
	CDVAE	0.835	0.602	0.962	0.976	0.962
	CDVAE-KNN	**0.842**	**0.623**	**0.966**	**0.984**	**0.970**

3.3.2 Generative Models Analysis.

We used the Parzen window method to estimate the likelihood of the synthesized samples following the distribution of the original data where a higher value presents a better generative model. Table 3.2 evidences that the quality of the generated data of CDAAE is always better than the other models. This result explains why machine learning algorithms trained on the augmented datasets of CDAAE often achieve better performance than those trained on the other

Algorithm 2 CDAAE-KNN algorithm.

INPUT:

X: original training set; $\tilde{X}$: generated data samples; m: number of nearest neighbours; d(x,y): Euclide distance between vector x and vector y

OUTPUT:

X_{new}: new sampling set

BEGIN:

1. Training CDAAE on X to have the trained decoder network (De)
2. Set $\tilde{X}$ contains minority samples generated by De
3. Run KNN algorithm on set $X \cup \tilde{X}$

for each $x_i \in \tilde{X}$ **do**

- Set m = number of majority class samples in k nearest neighbours
- Set n = number of minority class samples in k nearest neighbours

if $m \geq \alpha_1$ and $n \geq \alpha_2$ **then**

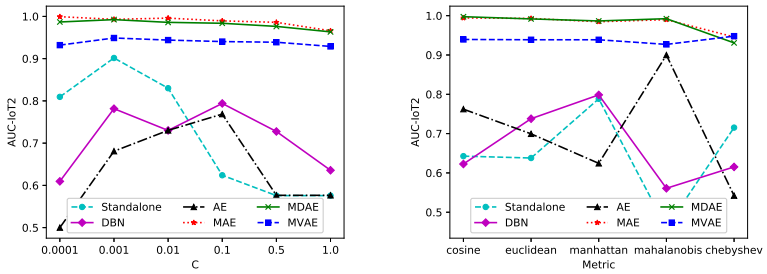
$X_{new} = X \cup \{x_i\}$

end if

end for

return X_{new}

END.



(a) SVM. (b) NCT.

Fig. 2.3: AUCs of (a) SVM and (b) NCT with different parameters on IoT-2.

2.3.4 Complexity of Proposed Models

In general, model complexity can be defined as a function of a number of trainable parameters. Table 2.3 presents the training time and the number of trainable parameters in each AE-based model. As presented in the training time and the trainable parameters of AE-based models with the same architecture are similar. Therefore, adjusting loss functions of the proposed models does not affect much to training time and model size of AE-based models.

2.4 Conclusion

In this chapter, we have designed three novel AE based models for learning a new latent representation to enhance the accuracy in NAD. In our models, normal data and known attacks are projected into two narrow separated regions in the latent feature space. To obtain such a latent representation, we have added new regularized terms to three AE versions, resulting in three regularized models, namely the MVAE, MAE and MDAE. These regularized AEs are trained on the normal data and known IoT attacks, and the bottleneck layer of the trained AEs was then used as the new feature space for linear classifiers.

Table 2.1: Comparision ofAUC scores.

Class-ifiers	Models	Datasets								
		IoT-1	IoT-2	IoT-3	IoT-4	IoT-5	IoT-6	IoT-7	IoT-8	IoT-9
RF	STA	0.979	0.963	0.962	0.670	0.978	0.916	0.999	0.968	0.838
SVM	STA	0.839	0.793	0.842	0.831	0.809	0.934	0.999	0.787	0.799
	DBN	0.775	0.798	0.950	0.941	0.977	0.822	0.960	0.772	0.757
	CNN	0.500	0.500	0.702	0.878	0.815	0.640	0.996	0.809	0.845
	AE	0.845	0.899	0.548	0.959	0.977	0.766	0.976	0.820	**0.997**
	VAE	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500
	DAE	0.849	0.990	0.569	0.968	0.980	0.803	0.982	0.818	0.996
	MVAE	0.914	0.948	0.978	0.985	0.932	0.950	**0.998**	0.826	0.858
	MAE	**0.999**	0.997	**0.999**	0.987	**0.982**	**0.999**	**0.999**	0.846	0.842
	MDAE	**0.999**	**0.998**	0.999	**0.992**	**0.982**	**0.999**	**0.999**	**0.892**	0.902
PCT	STA	0.768	0.834	0.568	0.835	0.809	0.933	0.998	0.753	0.802
	DBN	0.995	0.786	0.973	0.954	0.697	0.847	0.957	0.783	0.755
	CNN	0.500	0.500	0.674	0.877	0.812	0.635	0.996	0.797	0.844
	AE	0.849	0.892	0.498	0.965	0.977	0.813	0.977	0.814	0.815
	VAE	0.503	0.501	0.499	0.501	0.507	0.497	0.500	0.500	0.499
	DAE	0.882	0.903	0.534	0.969	0.982	0.862	0.984	0.857	0.849
	MVAE	0.954	0.947	0.972	0.986	0.923	0.923	0.997	0.823	0.849
	MAE	**0.996**	0.996	**0.999**	**0.998**	**0.989**	**0.999**	**0.999**	0.833	**0.991**
	MDAE	**0.996**	**0.997**	**0.999**	**0.998**	**0.989**	**0.999**	**0.999**	**0.889**	**0.991**
NCT	STA	0.743	0.747	0.498	0.785	0.692	0.570	0.993	0.770	0.748
	DBN	0.994	0.786	0.954	0.938	0.961	0.927	0.859	0.781	**0.964**
	CNN	0.500	0.500	0.680	0.877	0.767	0.632	0.977	0.777	0.799
	AE	0.985	0.767	0.498	0.834	0.835	0.997	0.945	0.746	0.767
	VAE	0.501	0.506	0.511	0.487	0.499	0.505	0.500	0.488	0.479
	DAE	0.989	0.770	0.580	0.882	0.863	0.997	0.966	0.806	0.788
	MVAE	0.846	0.939	0.973	0.984	0.927	0.937	0.998	0.822	0.796
	MAE	**0.998**	0.996	**0.999**	0.987	0.982	**0.999**	**0.999**	0.828	0.799
	MDAE	0.996	**0.998**	0.998	**0.992**	**0.985**	**0.999**	**0.999**	**0.887**	0.889
LR	STA	0.862	0.837	0.565	0.829	0.802	0.932	0.998	0.791	0.800
	DBN	0.776	0.939	0.960	0.955	0.961	0.837	0.962	0.779	0.755
	CNN	0.500	0.500	0.710	0.878	0.811	0.636	0.997	0.801	0.843
	AE	0.850	0.894	0.498	0.958	**0.987**	0.743	0.996	0.795	**0.998**
	VAE	0.500	0.499	0.500	0.500	0.500	0.500	0.500	0.500	0.500
	DAE	0.871	0.902	0.587	0.966	0.982	0.801	0.996	0.810	0.988
	MVAE	0.921	0.989	0.981	0.985	0.933	0.955	**0.999**	0.828	0.858
	MAE	**0.999**	0.997	**0.999**	0.988	0.984	**0.999**	**0.999**	0.835	0.840
	MDAE	0.996	**0.998**	0.998	**0.992**	0.985	**0.999**	**0.999**	**0.887**	0.889

$$G_{loss} = \frac{1}{n}\sum_{i=1}^{n}\log(d^i_{fake}). \tag{3.7}$$

The total loss function of CDAAE is the combination of the three above loss as in (3.8), and CDAAE is trained to minimize this loss function.

$$L_{CDAAE} = R_{loss} + D_{loss} - G_{loss}. \tag{3.8}$$

3.2.3 Borderline Sampling with CDAAE-KNN

In ACGAN-SVM, SVMs are also used to learn the boundary of classes and over-sampling the minor samples around these boundaries. However, when dealing with imbalanced datasets, the class boundary learned by SVMs may be skewed toward the minority class, thereby increasing the misclassified rate of the minority class. Therefore, we further improve choosing the borderline samples using the K nearest neighbor algorithm (KNN). Specifically, we propose a hybrid model between CDAAE and KNN (shorted as CDAAE-KNN) for generating malicious data for the NAD. In this model, KNN is used to select the generated samples by CDAAE that are close to the borderline.

Algorithm 2 describes the details of CDAAE-KNN. The algorithm divides into two phases: generation and selection. In the generation phase (Step 1 and 2), we train a CDAAE model on the original dataset X. We then use the decoder network De of CDAAE to generate new samples for the minority classes. The generated dataset is called $\tilde{X}$. In the selection phase, the KNN algorithm is executed in the sample set $X \cup \tilde{X}$ to find the k nearest neighbors of the samples $x_i \in \tilde{X}$. For each x_i, we calculate the number of nearest samples which belong to minority classes n and the number of nearest samples that belong to majority classes m. If m and n are larger than thresholds α_1 and α_2, respectively, the sample x_i will be considered as the borderline samples, and it is added to the output set X_{new}.

3.3 Results and Discussions

3.3.1 Performance Comparison

Overall, Table 3.1 shows that the generative models can be used to generate meaningful samples for the minor classes on NAD. Moreover, our proposed models often achieve better results compared to the previous models.

the bottleneck $\tilde{\mathbf{z}}$. The latent representation $\tilde{\mathbf{z}}$ is used to guide the decoder of CDAAE to reconstruct the input at its output from a desired distribution, i.e., the standard normal distribution. Let $\mathbf{W_{En}}$, $\mathbf{W_{De}}$, $\mathbf{b_{En}}$, and $\mathbf{b_{De}}$ be the weight matrices and the bias vectors of En and De, respectively, and $\mathbf{x} = \{x^1, x^2, \ldots, x^n\}$ be a training dataset where n is the number of training data samples, the computations of En and De based on each training sample are presented in Eq. 3.1 and Eq. 3.2, respectively.

$$\tilde{z}^i = f_{En}(\mathbf{W_{En}}(x^i_{noise} \mid y^t + \mathbf{b_{En}}), \tag{3.1}$$

$$\tilde{x}^i = f_{De}(\mathbf{W_{De}}(z^i \mid y^t) + \mathbf{b_{De}}), \tag{3.2}$$

where $\mid$ is concatenation operator and f_{En} and f_{De} are the activation functions of the encoder and the decoder, respectively. y^t is the label of the data sample x^i, x^i_{noise} is generated from x^i by a Gaussian noise. The reconstruction phase aims to reconstruct the original data $\mathbf{x}$ from the corrupted version $\mathbf{x_{noise}}$ by minimizing the reconstruction error (R_{loss}) in Eq. 3.3.

$$R_{loss} = \frac{1}{n}\sum_{i=0}^{n}\left(x^i - \hat{x}^i\right)^2. \tag{3.3}$$

In the regularization phase, an adversarial network is used to regularize the hidden representation $\tilde{\mathbf{z}}$ of CDAAE. The generator (Ge) of the adversarial network, which is also the encoder of the Denoising AutoEncoder (En), tries to generate the latent variable $\tilde{\mathbf{z}}$ that is similar to sample $\mathbf{z}$ drawn from a standard normal distribution, $p(\mathbf{z}) = \mathbb{N}(\mathbf{z}|0, 1)$. We define d_{real} and d_{fake} as the outputs of Di with inputs $\mathbf{z}$ and $\tilde{\mathbf{z}}$, respectively. Let $\mathbf{W_{Di}}$ and $\mathbf{b_{Di}}$ be the weight matrix and the bias vector of Di, respectively. For each data point x^i, d^i_{real} and d^i_{fake} are calculated as follows:

$$d^i_{real} = f_{Di}(\mathbf{W_{Di}}z^i + \mathbf{b_{Di}}), \tag{3.4}$$

$$d^i_{fake} = f_{Di}(\mathbf{W_{Di}}\tilde{z}^i + \mathbf{b_{Di}}), \tag{3.5}$$

where f_{Di} is the activation function of Di.

The discriminator (Di) attempts to distinguish the true distribution $\mathbf{z} \sim p(\mathbf{z})$ from the latent variable $\tilde{\mathbf{z}}$ by minimizing in Eq. 3.6 whereas the generator Ge (or En) tries to generate $\tilde{\mathbf{z}}$ to fool the discriminator Di by maximizing in Eq. 3.7.

$$D_{loss} = -\frac{1}{n}\sum_{i=0}^{n}\left(\log(d^i_{real}) + \log(1 - d^i_{fake})\right), \tag{3.6}$$

Table 2.2: AUC score of the NCT classifier on the IoT-2 dataset in the cross-datasets experiment.

Train/ Test botnets	**STA**	**DBN**	**AE**	**MVAE**	**MAE**	**MDAE**
Gafgyt/Mirai	0.747	0.732	0.717	0.943	0.974	0.988
Mirai/Gafgyt	0.747	0.720	0.628	0.999	0.999	0.999

Table 2.3: Complexity of AE-based models trained on the IoT-1 dataset.

Models	**Training Time**	**Trainable Parameters**
AE	370.868	25117
VAE	420.969	25179
DAE	405.188	25117
MVAE	408.047	25179
MAE	354.990	25117
MDAE	424.166	25117

Chapter 3

DEEP GENERATIVE LEARNING MODELS FOR NETWORK ATTACK DETECTION

3.1 Introduction

In Chapter 2, the proposed representation learning method helps to improve the accuracy of machine learning in NAD. However, this approach only performs well with the assumption that we can collect enough labeled data from both normal traffic and anomalous traffic. Nevertheless, many network attack datasets are imbalanced. In this chapter, we develop three deep generative models to synthesize malicious samples on the network systems. The proposed models generate samples that further improve NAD's accuracy.

3.2 Deep Generative Models for NAD

3.2.1 Generating Synthesized Attacks using Auxiliary Conditional Generative Adversarial Network-Support Vector Machine (ACGAN-SVM)

The first method for generating artificial data is ACGAN-SVM. ACGAN-SVM attempts to produce samples that are nearby the borderline area defined by the SVM model.

Algorithm 1 presents a detailed description of using ACGAN-SVM for generating synthesized data. The technique is divided into two main phases, i.e., generation and selection. In the generation phase, the ACGAN network is trained on the training dataset X. After that, the generator network (G) of ACGAN is used to generate synthesized samples X'. In the selection phase, the SVM model is trained on the training dataset X, and the set of support vectors of this model is called SV_s. For each support vector $sv_i \subset SV_s$, we calculate the average Euclidean distance d_i of m nearest neighbor samples of sv_i in X to sv_i. If a generated sample $x_j \subset X'$ has the distance to sv_i smaller than d_i, this sample is kept. Conversely, if the distance from x_i to sv_i is greater than d_i, the sample is removed. The algorithm will stop when the augmented dataset is balanced for every class. The augmented dataset is then used to train three classification algorithms as in ACGAN.

Algorithm 1 ACGAN-SVM algorithm for oversampling.

Inputs: original training set X, generated data samples X', number of nearest neighbors m, Euclid distance between vector x and vector y d(x,y).

Outputs: new sampling set X_{new}.

begin

Training ACGAN on X to have trained Generator G Set X' contains minority samples generated by G network Train SVM model on X to have the set of support vectors SVs

foreach $sv_i \in SVs$ **do**

Compute m nearest neighbors in X Compute d_i that is average of Euclid distance from m nearest neighbors to sv_i

foreach $x_j \in X'$ **do**

if $d(x_j, sv_i) \leq d_i$ **then**

$X_{new} = X \cup \{x_j\}$

return X_{new}.

3.2.2 Conditional Denoising Adversarial AutoEncoder (CDAAE)

The disadvantage of ACGAN-based model is that the training process is difficult to convergence. Thus, we propose a new generative model, i.e., CDAAE and CDAAE-K Nearest Neighbor (CDAAE-KNN) to improve NAD.

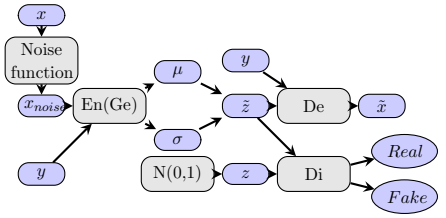


Fig. 3.1: Structure of CDAAE.

As described in Fig. 3.1, two networks in CDAAE are jointly trained in two phases, i.e., the reconstruction phase and the regularization phase. In the reconstruction phase, the encoder of CDAAE (En), receives two inputs, i.e., a noisy data $\mathbf{x_{noise}}$ and a class label $\mathbf{y}$, and generates the latent representation or