Hindawi
Security and Communication Networks
Volume 2023, Article ID 3965245, 10 pages
https://doi.org/10.1155/2023/3965245

Research Article

Embedded Gateway Security Detection Technology Based on the Deep Neural Network Rule Extraction

Jianming Shi and Tao Feng

School of Computer and Communication, Lanzhou University of Technology, Lanzhou 730050, China

Correspondence should be addressed to Tao Feng; fengt@lut.edu.cn

Received 5 July 2022; Revised 22 February 2023; Accepted 11 March 2023; Published 25 April 2023

Academic Editor: Yu Yao

Aiming at the network security problem of power system cable trench control industrial Internet system, we studied an intrusion detection method applied to the embedded industrial Internet of Things gateway. This method extracts rules from the DBN-DNN deep neural network to obtain intrusion detection models that are conducive to integration into embedded systems. We first use the DBN network to reduce the dimensionality of the data, then use the DNN to train the classification model, and extract the rules from the DNN's neurons to form a rule tree for intrusion detection. The KDD CUP99 training database is used to verify the feasibility of the method, and the test is carried out in the embedded gateway. The results show that the detection method based on rule extraction used in this paper can ensure detection efficiency and accuracy compared to the traditional detection methods. At the same time, it saves more computing resources and is more conducive to integration in embedded gateway systems.

1. Introduction

With the advent of the information age, the application of the industrial Internet has become increasingly popular, and increased security issues have followed [1]. The wide application of network interconnections in production and manufacturing not only brings convenience and efficiency to production, but also many hidden dangers to security. At present, the security of the Industrial Internet mainly involves six major security issues: equipment security, control security, network security, identification resolution security, platform security, and data security [2]. Among them, the security of platforms and data is a new issue that needs to be urgently addressed for industrial Internet security. Once a platform and data are compromised and attacked, it can cause serious damage and loss to the entire production system. Intrusion detection is an effective network attack detection method, which is widely used in the field of network security detection [3]. The traditional intrusion detection technology is divided into misuse-based intrusion detection technology (MIDS) and anomaly-based intrusion detection technology (AIDS). Among them, intrusion detection technologies related to machine learning include an intrusion detection technology based on autoencoder [4], intrusion detection technology based on deep learning [5–7], intrusion detection technology based on reinforcement learning [8], and an intrusion detection technology based on visual analysis [9]. Detection methods based on deep learning can process large-scale network traffic data more effectively, and have a higher detection efficiency and accuracy [10]. The performance of DBN-based detection technology in network intrusion detection is widely recognized by researchers, but due to the complex training process of deep neural networks, poor model interpretability, and high requirements for equipment computing power [11], industrial Internet security is different from traditional Internet security and industrial Internet is a real-time requirement closely related to the production. In the network, if there is a problem, it will not be solved in time, which will cause heavy losses [12]. The breadth of the Industrial Internet determines that it will not be equipped with Internet gateways with high computing power, regardless of economic or practical considerations, which makes it difficult

for traditional network detection methods to be directly applicable to complex industrial Internet systems [13]. There are also some research studies on the security detection methods for embedded systems. Some of them study the optimization of embedded system architecture and try to optimize the security detection performance by adding complex detection devices, which considerably increases the deployment cost of embedded systems. The other part is to study the optimization of energy consumption and memory utilization in the implementation of security detection methods for embedded systems, which can achieve better security detection results by saving computational energy consumption and using the limited memory cells of the embedded systems efficiently. Despite the impact of these studies, most of them are still at the theoretical stage and there are still some issues to overcome in the actual deployment of the embedded systems. Reference [14] uses a method to detect malicious network traffic by extracting learning rules from neural networks. This method reduces the computational requirements during the detection process, and may be more beneficial to devices with small computational resources. In the current research study on industrial network security, it is very necessary to study the security detection methods that are more conducive to the deployment of embedded systems. Based on existing research studies, this study studies a security detection method that is more conducive to implementation in the embedded systems. The contribution of this research is as follows:

(i) Aiming at the application of traditional security detection technology in the embedded gateway of industrial Internet, due to the limited hardware computing resources, difficulties in deployment, and weak detection capabilities, this paper proposes a solution to save computing resources while ensuring the detection effect.

(ii) In order to make the security detection technology easier to deploy to the embedded gateway, this paper proposes a detection method that first extracts rules from the DBN-DNN network, and then deploys the rules to the embedded gateway, and designs an experimental scheme to verify it.

(iii) The detection technology based on the deep network rule extraction is tested on the KDD CUP99 dataset, which proves the feasibility of the method. The results show that the method guarantees an excellent detection effect when converting complex neural networks into logical calculations that is more conducive to the implementation in embedded systems.

2. Related Work

Existing research studies show that the machine learning methods have been widely used in the study of security detection methods as machine learning techniques have been developed. This section introduces the work related to the method studied in this paper in the following two parts: the security detection method based on machine learning and the security detection method of embedded IoT gateway with limited resources.

2.1. Safety Detection Method Based on Machine Learning. Machine learning-based security detection methods include artificial neural networks, association rules, and fuzzy association rules, Bayesian networks, clustering, decision trees, integrated learning, evolutionary computing, hidden Markov models, inductive learning, Naive Bayesian, sequential pattern mining, and support vector machines. Reference [15] proposes a network intrusion detection method based on deep learning. The deep confidence neural network is used to extract the features of the network monitoring data, and the BP neural network is used as the top-level classifier to classify the intrusion types. The verification results show that this method is significantly improved compared with the traditional machine learning methods. Reference [16] proposes an integration method to improve the detection performance by simultaneously constructing numerous independent decision trees for different subsets in different parts of the training samples. This method improves the accuracy of the method by combining numerous decision trees for final judgment because any decision tree in the random forest is different, and the variance is reduced, which makes the method have a strong generalization to avoid problems such as dataset imbalance and overfitting. Reference [17] proposes a hybrid detection framework that depends on data mining classification and clustering technology. The random forest classification algorithm is used to automatically build intrusion patterns from the training dataset, and then the K-Means clustering algorithm is used to detect new intrusions. This method detects the intrusion of one or more clusters by clustering the detection data. Reference [18] proposed a hybrid network-based high-efficiency model (HNIDS) using the enhanced genetic algorithm, particle swarm optimization (EGA-PSO), and the improved random forest (IRF) method. This method uses the mixed EGA-PSO method to enhance the secondary sample. By adding multiobjective functions to select the best features and to achieve improved fitness results, the decision tree list is merged in each iteration process, thereby effectively preventing the overfitting problem.

2.2. Security Detection Methods for Embedded Systems. Research studies on security detection techniques for industrial Internet-embedded systems have also yielded various results in the recent years. Reference [19] proposed a multicore-based detection architecture for real-time embedded systems, which is related to a novel monitoring technology. The security of the real-time embedded system can be improved by analyzing and observing the inherent property of the real-time system, and detecting malicious activities through statistical analysis of its execution. Reference [20] analyzes the detailed energy of the feature extraction engine and the three machine learning classifiers are implemented in decision tree (DT), Naive Bayes (NB), and k-nearest neighbors (KNN) in the embedded system security

detection technology. It hopes to propose more energy-saving detection methods. Reference [21] proposes a detection technology optimized for embedded system memory, which maximizes the coverage of security attributes relative to available memory, and can be applied to various embedded devices with different memory capacities. It provides a strong detection rate even when memory is limited. Reference [22] proposes an integrated security detection method for Internet of Things (IoT) devices, which combines multiple classifiers to find an accurate classifier. An integrated classification model using automatic model selection is proposed. The model uses a large number of classifiers with different configurations, and the model is evaluated and selected by an ensemble metric. Reference [23] proposes an intrusion detection system based on machine learning to detect the IoT network attacks. A new layered intrusion detection system is proposed for the backbone network of the IoT using a two-layer dimension reduction engine and a two-layer classification engine. After dimensionality reduction via component analysis and linear discriminant analysis units, a Naive Bayesian classifier is used to classify the attack records via the k-nearest algorithm. Experiments show that the proposed method has excellent detection performance in hard-to-capture attacks. While these methods have improved the detection methods, to a certain extent, by making them more suitable for embedded systems, they have some shortcomings in terms of detection accuracy and efficiency.

3. Method Principle

Embedded networks have always occupied a large proportion of the current industrial Internet applications. As an essential information forwarding unit in the embedded networks, embedded gateways are the main equipment to ensure the embedded network security. However, from the perspective of cost, most embedded gateways are not equipped with high computing power, which makes it difficult for the traditional security detection technology to obtain a perfect representation of the embedded gateways. It is of great significance for the development of the industrial Internet to find a detection technology that can ensure the detection efficiency and is more conducive to integration into the embedded gateway. In the traditional detection technology, the intrusion detection technology based on the deep neural network has a better detection performance, but because of the complex structure of the deep network, poor interpretability of the detection process, and the high computing requirements, it is difficult to adapt to embedded gateways. To improve the real-time performance of the detection technology and to facilitate the transplantation of embedded gateways, a detection technology based on the DBN-DNN network rule extraction is proposed. First, a high-precision detection model is trained by the DBN-DNN network, and then the idea of the decision trees is used to extract the detection rules from the DBN-DNN network to obtain a rule model that is easier to integrate into the embedded gateway. The specific process in the method is shown in Figure 1.

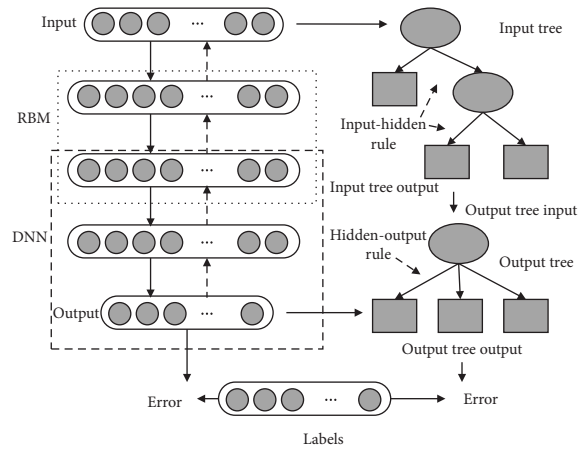

Figure 1: Detection model based on DBN-DNN rule extraction.

4. Detection Model Based on DBN-DNN

4.1. DBN-DNN Neural Network. First, we build a deep neural network training intrusion detection model based on the DBN-DNN structure. DBN is a kind of restricted Boltzmann machine (RBM) composed of multiple generative neural network structures, which is trained by the contrastive dispersion algorithm (CD) [24]. The extraction and dimensionality reduction of the RBM has excellent learning ability. The RBM training process is shown in Figure 2.

In the figure, h is the value vector of the neurons in the hidden layer, v is the value vector of the neurons in the visible layer, c is the paranoid vector of the hidden layer, b is the bias vector of the visible layer, and W is the weight matrix, which can connect the function to the given state. The probability distributions of v and h are shown in formulas (1) and (2), respectively.

$$E(v,h) = -c^T v - b^T h - h^T W v, \tag{1}$$

$$P(v,h) = \frac{1}{Z} e^{-E(v,h)}, \qquad P(v \mid h) = \frac{P(v,h)}{P(v)}. \tag{2}$$

In the formula, Z is the normalization coefficient, and the DNN network layer is added after the last layer of the RBM. The DNN layer takes the dimensionality reduction feature output by the RBM as the input vector, and uses the backpropagation algorithm (BP) for fine-tuning and for supervised training entity relationship classifier. This network structure is called the DBN-DNN network structure [25], and the network structure is shown in Figure 3.

4.2. Feature Dimensionality Reduction Based on DBN. There is a training dataset $D = ((X1, Y1), (X2, Y2) \ldots (Xn, Yn))$, where n is the number of samples in the dataset, and if each sample inputs h features, then X_n^h represents the h-th feature of the n test data samples. We build an m-layer RBM network to reduce the dimensionality of the input sample

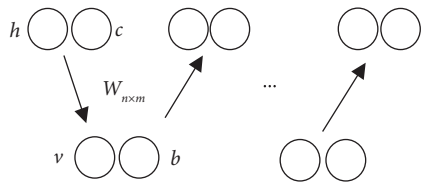

Figure 2: RBM training process.

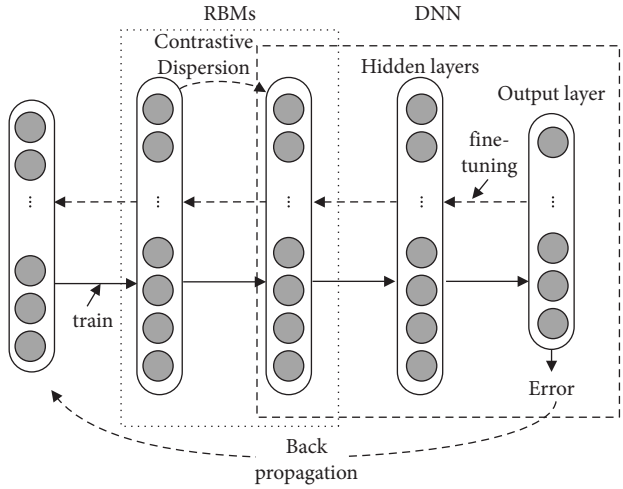

Figure 3: DBN-DNN network structure.

features, let the number of features after RBM dimensionality reduction be $h0$, and we use equation (3) to initialize the number of feature vectors of the RBM output layer, and then update the output of the $(m-1)$-th layer RBM according to equation (4), that is, the numbers of features.

$$h0 = \frac{1}{a} \times h, \tag{3}$$

$$h(m-1) = h0 + H\left(\frac{h - h0}{m}\right). \tag{4}$$

In equation (3), a is the feature dimensionality reduction ratio, in equation (4), $H(\,)$ is the rounding function, and so on to update the number of output features of each RBM layer. In order to obtain a better feature dimensionality reduction effect, the particle swarm algorithm is used to optimize the RBM layer structure, and a fitness evaluation function based on the average reconstruction error of the dimensionality reduction model is constructed to optimize the number of RBM layers.

4.3. DNN-Based Entity Classifier. After optimization, the number of output features of the m-th layer RBM is $h0$, and the output of the m-layer RBM is used as input to build a DNN entity classifier. The input layer of the classifier has h0 neurons, and the output layer is the corresponding detection type. Using the ReLU function as the activation function, it adopts the backpropagation algorithm to adjust the weight, and the activation function is shown in equation (5):

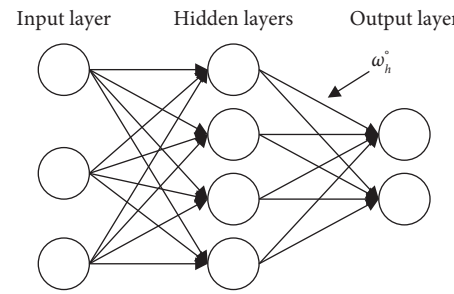

Figure 4: Schematic diagram of the structure of the DNN entity classifier.

$$f(x) = \max(0, x). \tag{5}$$

In the formula, when the input is less than 0, the output is 0, and when the input is greater than 0, the output is the same as the input. The loss function is selected as shown in equation (6):

$$J(W, b, x, y) = \frac{1}{2}\left\|a^L - y\right\|_2^2, \tag{6}$$

where a^L and y are vectors with the same feature dimension as the output, and $\|a^L - y\|_2$ is the L2 norm of $a^L - y$. After selecting the loss function, the gradient descent method is used to iteratively obtain the weights of each layer. The structure of the DNN entity classifier is shown in Figure 4.

5. DBN-DNN Network Rule Extraction

Extracting rules from DBN-DNN is divided into three steps. First, we obtain the neuron output of the hidden layer, then extract the hidden layer to the output layer to generate a rule tree between the input layer and each hidden layer. Finally, since the output of the input rule tree is the input of the output rule tree, the output of the input rule tree is used as the input of the output rule tree to construct a complete rule tree detection model.

5.1. Decision Tree-Based Rule Extraction. The rule extraction method is based on the output of each hidden layer neuron. We assume that the output feature of the RBM layer is X, the detection result is Y, and the number of hidden layers of the DNN is j. Each hidden layer contains k neurons, where h_j^k denotes the k-th neuron in the j-th hidden layer. X_i is a sample of the dataset, and Y_i is the detection type corresponding to sample X_i. Assuming that X_i has n eigenvalues, the output of the k-th neuron in the j-th hidden layer is $O(h_j^k)$ as shown in equation (7):

$$O\left(h_j^k\right) = \mathrm{Re}LU\left(\sum_{m=1}^{n_{j-1}} w_m x_m - \theta\right), \tag{7}$$

where w_m is the connection weight of the m-th neuron and n_{j-1} is the number of neurons in the hidden layer of $j-1$. x_m is the input of the m-th neuron, and θ is the threshold. We then calculate the output mean O_j of the j-th hidden layer neuron as shown in equation (8):

$$O_j = \frac{1}{n_j}\sum_{i=1}^{n_j} O\left(h_j^i\right), \tag{8}$$

where n_j is the number of neurons in the j-th hidden layer. At this time, the average value of neurons in each hidden layer corresponding to X_i can be obtained, and these average values can be used to establish decision rules.

5.2. Input Rule Tree Model. The function of the input rule tree is to extract and describe the rules between the input features and the hidden layer of the neural network. The number of input rule trees depends on the output features of the m-th layer of the RBM and is equal to the number of hidden layers of the trained DNN. Let X_i be an m-dimensional feature vector, and the output mean of its corresponding j-th hidden layer is O_j^i , and the k-th variable x_i^k in X_i is used as the segmentation variable and the segmentation point, and the defined regions R_1 and R_2 are shown in equations (9) and (10):

$$R_1\left(k, x_i^k\right) = \left\{x \,\middle|\, x^k \le x_i^k\right\}, \tag{9}$$

$$R_2\left(k, x_i^k\right) = \left\{x \,\middle|\, x^k > x_i^k\right\}. \tag{10}$$

Then, the optimal segmentation variable and segmentation point are obtained, and the optimal value is calculated as shown in equation (11):

$$\min_{k, x_i^k}\left[\sum_{X_i \in R_1\left(k, x_i^k\right)}\left(O_j^i - c_1\right) + \sum_{X_i \in R_2\left(k, x_i^k\right)}\left(O_j^i - c_2\right)\right]. \tag{11}$$

In the formula, c_1 and c_2 are the output values of the two regions after division, which are the values with the smallest square error in the respective regions. The calculation process is shown in equations (12) and (13):

$$c_1 = \frac{1}{N_1}\sum_{X_i \in R_1\left(k, x_i^k\right)} O_j^i, \tag{12}$$

$$c_2 = \frac{1}{N_2}\sum_{X_i \in R_2\left(k, x_i^k\right)} O_j^i. \tag{13}$$

In the formula, N_1 is the number of samples divided into the region $R_1(k, x_i^k)$, and N_2 is the number of samples divided into the region $R_2(k, x_i^k)$. After finding the optimal segmentation point, the input space is divided into two regions in turn, and then the abovementioned division is repeated for each region. The process is repeated until the stopping condition is met, and finally a least squares regression tree is generated.

5.3. Output Rule Tree Model. The function of the output rule tree is to extract and describe the rules between the hidden layer of the neural network and the output of the neural network. After obtaining the output mean of the hidden layer neurons, the decision tree is used to establish the rules between the hidden layer mean and the output detection type. First, the empirical entropy $H(O, Y)$ of the hidden layer mean of the sample and the output detection type is calculated, let the output mean vector of the hidden layer corresponding to X_i be O_i, O_i is a j-dimensional vector, and j is the set number of hidden layers. The calculation process is shown in equation (14):

$$H(O, Y) = -\sum_{i=1}^{n} pi \log pi. \tag{14}$$

In the formula, n represents the total number of types of detection results, pi represents the proportion of the i-th type of detection results, and log is the logarithm of base 2 or e. We use the dichotomy method to build a rule tree, set the total number of samples to be N, divide the output mean of the j-th hidden layer of all samples into O_j^+ and O_j^-, set the boundary value to O_j^t, and calculate the O_j^t as shown in equation (15):

$$O_j^t = \left\{\frac{O_j^i + O_j^{i+1}}{2} \mid 1 \le i \le N\right\}. \tag{15}$$

Then, the information gain of different O_j^t for different detection results of the dataset is calculated, respectively, and finally, the optimal demarcation point with the largest information gain is selected to establish the output rule tree model in turn.

5.4. Whole Rule Tree Model. After obtaining the input rule tree and the output rule tree through training, the input rule tree and the output rule tree are combined into a complete rule tree model. The input rule tree is used to describe the learning process between the input layer and the hidden layer of the neural network, and the output rule tree is used to describe the rules between the hidden layer and the output of the neural network. The detection result of the input rule tree is used as the input of the output rule tree to build the model. The number of input rule trees depends on the number of hidden layers of the training neural network. Each input rule tree is a description of the learning process between the input feature and a hidden layer of the neural network. After obtaining the description between the input feature and all hidden layers, the final detection result is obtained by outputting the rule tree as fresh data, and the input of the output rule tree is the output of all the input rule trees.

6. Experimental Design

In order to verify the feasibility of the proposed method, the KDD CUP99 dataset is used to design experiments to analyze the detection effect of the rule tree detection model. The original data of the KDD CUP99 dataset comes from the DARPA Intrusion Detection Evaluation Project in 1998. The dataset contains 500 10,000 training data and two million test data.

6.1. Experimental Environment and Implementation. The experimental environment is Windows 10 64 system, CPU frequency is 3.6 MHz, memory is 16G, graphics card is

GTX1050ti, and graphics card memory is 4G. The software environment includes the open-source machine learning platform TensorFlow and the free machine learning library scikit-learn. The experiment uses Python as the programming language. In the experiment, the data is first processed, and then a deep neural network model is built in TensorFlow to obtain the training process data. Then, a rule tree is built through scikit-learn to extract the rule descriptions of the learning process from the input layer to the hidden layer and from the hidden layer to the output layer from the deep neural network process data, and finally a complete detection rule tree is established.

6.2. Datasets and Data Preprocessing. Anomaly types in the KDD CUP99 dataset are subdivided into 4 categories with a total of 39 attack types [26], of which 22 attack types appear in the training set, and another 17 unknown attack types appear in the test set. In the experiment, the character features are converted into numerical features, and then the feature values are standardized. First, the average value and average absolute error of each attribute are obtained. Let the k-th attribute of the i-th sample be x_i^k, then the sample k-th and the mean value $\overline{x^k}$ of the attribute is calculated as shown in equation (16), and the mean absolute error S^k is calculated as shown in equation (17):

$$\overline{x^k} = \frac{1}{N}\sum_{i=1}^{N} x_i^k, \tag{16}$$

$$S^k = \frac{1}{N}\sum_{i=1}^{N}\left|x_i^k - \overline{x^k}\right|. \tag{17}$$

In the formula, N is the total number of samples, and the standard value of x_i^k after normalization is set as $x_i^{'k}$. The calculation process of $x_i^{'k}$ is shown in equation (18).

$$x_i^{'k} = \frac{x_i^k - \overline{x^k}}{S^k}. \tag{18}$$

During the calculation, if any one of $\overline{x^k}$ and S^k is 0, then the value of $x_i^{'k}$ is also 0. After obtaining the standard value $x_i^{'k}r$, the data is normalized. Let $x_i^{''k}$ be the value after $x_i^{'k}$ normalization. The calculation process of $x_i^{''k}$ is as shown in equation (19):

$$x_i^{''k} = \frac{x_i^{'k} - x_{\min}}{x_{\max} - x_{\min}}, \tag{19}$$

where $x_{\min}$ is the minimum value in $x_i^{'k}$, and $x_{\max}$ is the maximum value in $x_i^{'k}$. 10% of the data in the KDD CUP99 dataset was selected for the experiment, with a total of 494,021 sample records. Table 1 shows the mean absolute error (MAE), standard deviation (SD), skewness of the first 8 sample features in the dataset after the character features are digitized (SKEW), and kurtosis (KURT).

It can be seen from Table 1 that the distribution interval of the sample features is large, and the experimental data is standardized. Table 2 shows the parameters of the first 8 sample features in the dataset after the dataset is standardized.

Table 1: Characteristic parameter table of samples after digitization.

FVP	MAE	SD	SKEW	KURT
$X1$	94.7744	707.74	25.865	942.53
$X2$	0.9308	0.96	−0.385	−1.81
$X3$	13.1866	15.26	0.960	−0.88
$X4$	1.7744	2.25	−1.809	2.22
$X5$	4806.56	988218.10	699.213	490584.34
$X6$	1535.05	33040.00	136.759	20338.14
$X7$	0.0001	0.01	149.842	22450.72
$X8$	0.0128	0.14	21.719	476.09

Table 2: Standardized sample feature parameter table.

FVP	MAE	SD	SKEW	KURT
$X1$	0.002288	0.009955	−9.129	83.168
$X2$	0.002177	0.002601	−0.815	1.417
$X3$	0.000271	0.000338	0.582	−0.318
$X4$	0.001911	0.002787	2.693	6.828
$X5$	0.004687	0.005974	3.185	28.308
$X6$	0.009612	0.015493	−2.814	7.048
$X7$	0.002019	0.151258	149.842	22450.73
$X8$	0.007068	0.074267	21.726	476.37

It can be seen from Table 2 that after the sample features are standardized, the distribution interval is significantly reduced, and the standardized data is used for deep model training and rule extraction.

6.3. DBN-DNN Network Model. After the data standardization is completed, the standardized samples are used to build a DBN-DNN deep network training detection model for feature extraction.

6.3.1. DBN Structure Design. Different RBM layer designs have a certain influence on the extraction effect of the sample features during DBN model training. The particle swarm algorithm is used to optimize the number of DBN layers and the number of neurons in each layer to obtain the optimal model parameters. The feature dimensionality reduction ratio is set to 3 times, and the number of DBN layers is an integer between 2 and 4 to simplify the model training. The number of neurons in each layer is updated as shown in equation (20):

$$h(m-1) = h0 + H\left(\frac{h - \Delta h - h0}{m}\right), \tag{20}$$

where h is the total number of input sample features, Δh is the number of neurons, and the adjustment parameter ranges from 0 to $h/3$, m is the number of DBN layers, $h0$ is the number of output features after dimensionality reduction, and the numerical calculation of $h0$ is shown in equation (21):

$$h0 = \frac{h - \Delta h}{3}. \tag{21}$$

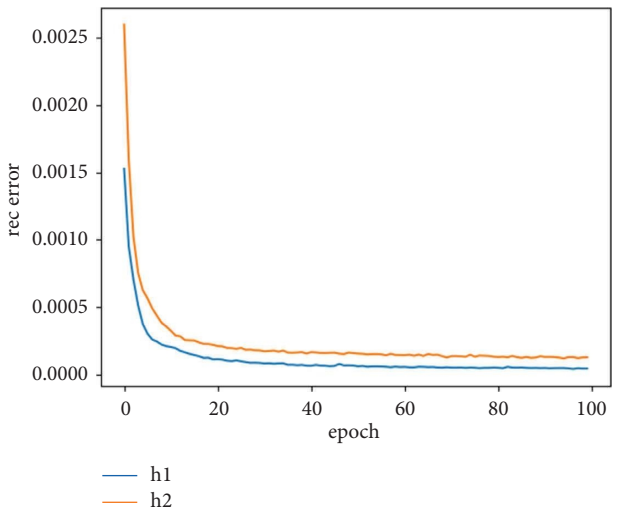

Figure 5: RBM layer reconstruction error.

The fitness update function is designed with the reconstruction error of each layer of the DBN, and the model parameters with the smallest mean reconstruction error are obtained by updating when m is set differently. After many experiments, when the number of RBM layers is set to 3 and the neuron adjustment parameter is set to 7, the DBN network has a better effect. At this time, the number of neurons in the input layer of the RBM is 34, and the number of neurons in the output layer is 12. Figure 5 shows the RBM reconstruction error from the input layer to the hidden layer and from the hidden layer to the output layer. During training, the parameters are updated every 300 samples, and the number of iterations is 100.

In the figure, $h1$ is the reconstruction error from the RBM input layer to the hidden layer, and $h2$ is the reconstruction error from the RBM hidden layer to the output layer. It can be seen from the figure that the reconstruction error is almost 0.0001 when the iteration reaches 40 times. At this time, the model has a better feature extraction effect.

6.3.2. DBN-DNN Model Training. In order to reduce the complexity of the rule extraction process in the experiment, a DNN network with one hidden layer is designed for the model training. Table 1 shows the total number of various attack identifiers in KDD CUP99, and the randomly selected ones from the different attack types during model training, that is, the number of training and testing samples.

As shown in Table 3, there are 18,000 training samples and 10,000 test samples. The number of iterations of the DNN network during training is set to 20. Figure 6 shows the results after 20 training sessions.

It can be seen from the figure that the model error is almost close to 0.001 after 20 sessions of training, and the trained model is saved for rule extraction.

6.4. Model Rule Extraction. After training the DBN-DNN model, a decision tree is built to extract the model rules. First, we calculate the output mean value of each hidden layer of the neural network, then establish an input rule tree that describes the characteristics of the input samples and the rules between the hidden layers of the neural network, and establish an output rule tree that describes the relationship between the hidden layer and the output of the neural network, and finally combine the input rule tree and the output rule tree to obtain a complete detection rule tree.

Table 3: DBN-DNN model training data table.

Attack type	Number	Train	Test
Normal	97278	4500	2500
DOS	391458	10500	6455
Probing	4107	2450	800
R2L	1126	500	200
U2R	52	50	45

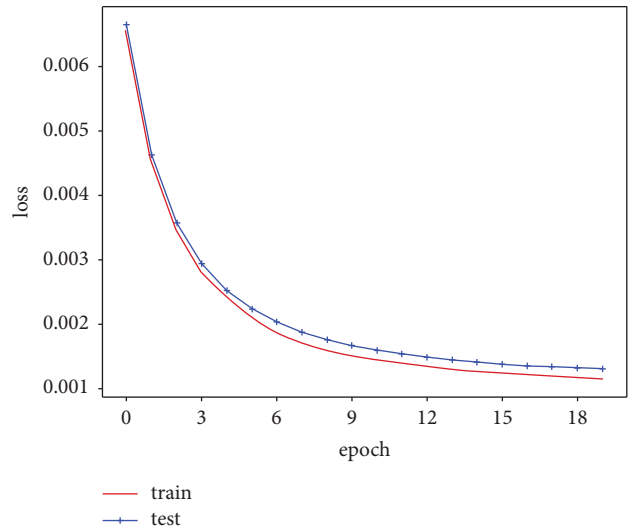

Figure 6: DBN-DNN network training error.

6.4.1. Building an Input Rule Tree. The output result is the feature of the neural network and the output mean of each hidden layer, and the output result is the predicted value of the output mean of each hidden layer corresponding to the input feature, so the input rule tree is a regression decision tree. The number of trees depends on the number of hidden layers of the deep neural network. Each rule tree uses the neural network input as the training sample for feature selection and division. Since the neural network selected in this experiment to reduce the complexity only contains one hidden layer, so only one input rule tree needs to be trained. The number of training samples is 18,000. Table 3 shows the sample types in detail. During training, we select the mean absolute error as the criterion for selecting features and splits. The input rules obtained after training the tree model are shown in Figure 7. It can be seen from the figure that all input samples have been divided, and the model selects the second, third, fourth, and eighth features of the input samples as the optimal segmentation features to establish a regression decision tree, and the leaf nodes of the tree are the hidden layer of the neural network corresponding to the

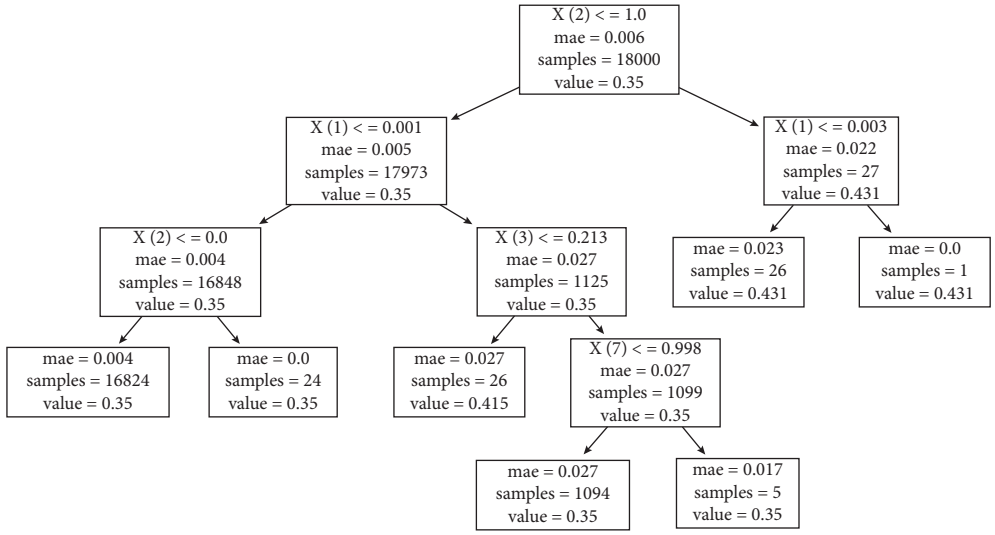

Figure 7: Input rule tree model.

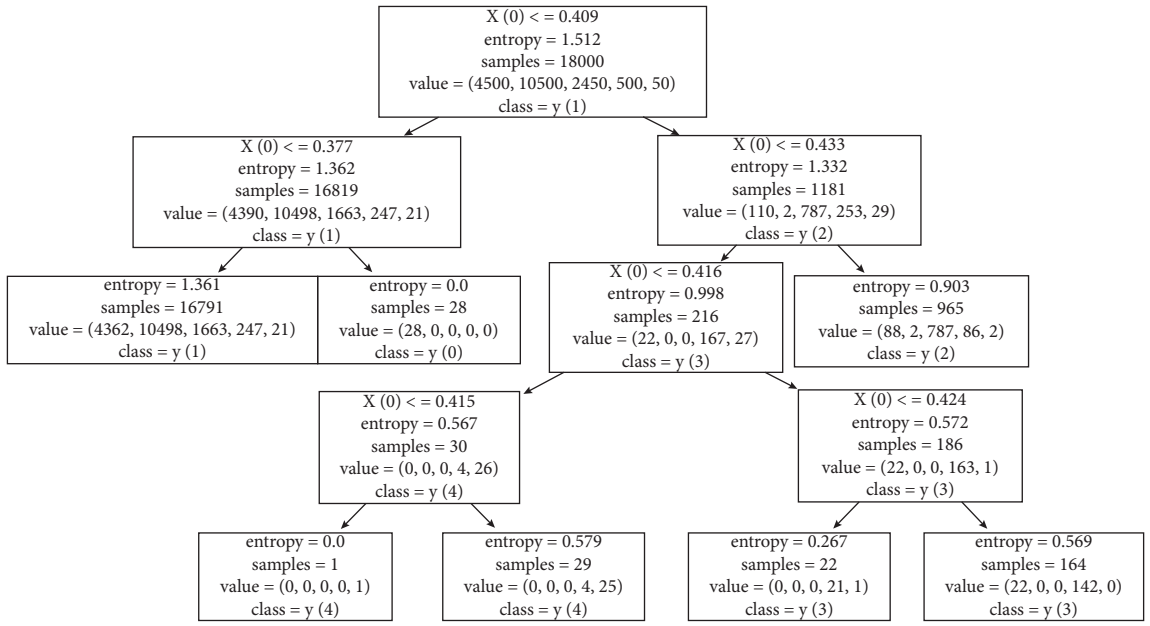

Figure 8: Output rule tree.

input sample that outputs the mean predicted value, and the output of the input rule tree is also used as the input value of the output rule tree.

6.4.2. Building an Output Rule Tree. The output rule tree is used to extract the rules between the hidden layer and the output layer of the deep neural network. The input of the output rule tree is the average output value of each hidden layer of the neural network corresponding to the sample, and the number of input values is equal to the number of hidden layers of the neural network. The output value of the output rule tree is the detection result of the deep neural network, so the output rule tree is a classification decision tree. Since the experiment uses a neural network with a hidden layer, the output rule tree contains only one input value. We take the output mean of the neural network hidden layer of the corresponding sample as the input sample, and use the neural network output detection type as the label to build a classification decision tree, select the information entropy as the measurement standard, and use the sample parameters listed in Table 3 to build the output rule tree, and the trained output rule tree is shown in Figure 8.

The leaf node of the rule tree in the figure is the five detection results of the classification, and it is also the output value of the deep neural network. After the output rule tree is established, the output of the input rule tree is used as the input of the output rule tree to establish a detection rule tree and then to verify the detection results.

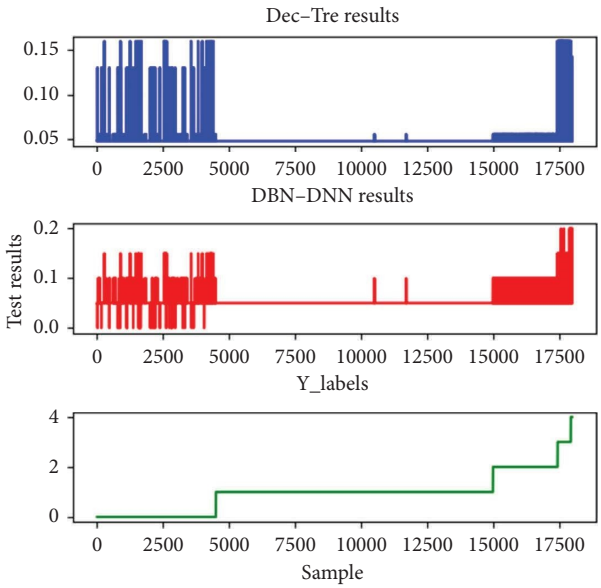

Figure 9: Comparison of detection effects.

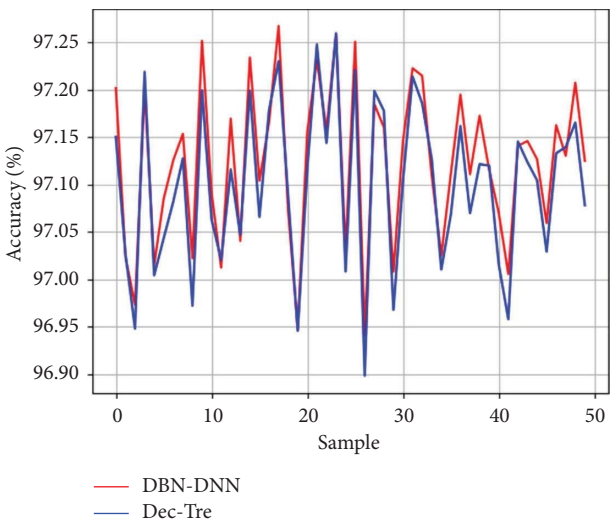

Figure 10: Accuracy comparison chart.

6.5. Experiment Analysis. To verify the detection effect of the rule tree, the deep neural network is used in the experiment and the established detection rule tree was used to detect the sample data. Each time, 18,000 samples were randomly selected from the experimental dataset according to the structure of the train in Table 3 as a group. A total of 50 groups were selected for the experiment. Figure 9 shows the detection results of a group of random data.

As can be seen from the figure, both the rule tree model and the deep neural network can better detect various attack types. Figure 10 shows the detection accuracy of the two methods.

It can be seen from the figure that the detection effects of the two detection methods are almost similar. To compare the detection effects of the deep neural network and the detection rule tree more clearly, the precision, recall, and the *F*-Measure of the 50 sets of test data were calculated, respectively. The mean values are compared, and the calculation results are shown in Table 4.

Table 4: Mean value table of test results of 50 groups of test data.

Method	Precision	Recall	*F*-measure
Dec-tree	0.9689	0.9814	0.9751
DBN-DNN	0.9691	0.9821	0.9756

From the table, it can be further concluded that the detection result of the rule detection model is almost close to that of the deep neural network. By combining Figure 10 and Table 4, it can be seen that the detection accuracy of the rule model is almost close to the detection accuracy of the neural network. At the same time, it is affected by the detection accuracy of the neural network. The detection effect of the detection model will vary with the detection effect of the neural network. Although its detection effect is slightly lower than that of the neural network, it converts complex mathematical operations into logical judgments that are easier to implement in the embedded systems and thus more conducive to real-time detection.

7. Conclusion

Aiming at the network security problems existing in the industrial Internet control system of substations, this paper proposes a security detection method for embedded industrial IoT gateways based on the deep neural network rule extraction. By extracting the rules of the DBN-DNN deep neural network model, as based on the rule tree security detection model, the model converts the complex calculation in the neural network into a logical judgment that is easier to implement in the embedded system, saves the detection cost while ensuring the detection accuracy, and in addition improves the detection efficiency. By using KDD CUP99 to verify, the results show that the detection effect of the detection method based on the rule extraction proposed in this paper is nearly close to the detection effect of the neural network of the extracted rules, and will improve with the improvement of the neural network detection effect. The extracted detection models are easier to understand and implement than the deep neural networks, and are more conducive to integration in the embedded systems. Subsequent work will be carried out to improve the detection accuracy of the deep neural networks. By comparing different network structures and training methods, a network model with higher accuracy will be obtained, and the rule extraction method will be continuously improved.

Data Availability

The dataset we used in this paper is available at https://https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Readers who are interested in our research can access the dataset and reproduce our results.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation for the Key Research and Development Program of Gansu Province, China (Grant no. 20YF3GA016).

References

[1] X. Yao, M. Liu, J. Zhang, T. Tao, H. Lan, and D. Ge, "Intelligent manufacturing from the perspective of AI: past, present and future," *Computer Integrated Manufacturing System*, vol. 25, no. 1, p. 16, 2019.

[2] R. K. Vigneswaran, R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Evaluating shallow and deep neural networks for network intrusion detection systems in cyber security," in *Proceedings of the 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp. 1–6, IEEE, Bengaluru, India, July 2018.

[3] D. U. Case, "Analysis of the cyber attack on the Ukrainian power grid," *Electricity Information Sharing and Analysis Center (E-ISAC)*, vol. 388, pp. 1–29, 2016.

[4] X. Qi and W. Xu, "Botnet detection model based on noise reduction automatic encoder [J]," *Journal of China University of Metrology*, vol. 30, no. 2, pp. 203–209, 2019.

[5] K. Jin, N. Shin, J. Seung Yeon, and S. H. Kim, "Method of intrusion detection using deep neural network," in *Proceedings of the 2017 IEEE international conference on big data and smart computing (BigComp)*, pp. 313–316, IEEE, Jeju, Feburary 2017.

[6] M. Ramaiah, V. Chandrasekaran, V. Ravi, and N. Kumar, "An intrusion detection system using optimized deep neural network architecture," *Transactions on Emerging Telecommunications Technologies*, vol. 32, p. 4221, 2021.

[7] R. Lohiya and A. Thakkar, "Intrusion detection using deep neural network with antirectifier layer," in *Applied Soft Computing and Communication Networks*, Springer, Berlin, Germany, 2021

[8] Q.-V. Dang and T.-H. Vo, "Studying the Reinforcement Learning techniques for the problem of intrusion detection," in *Proceedings of the 2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pp. 87–91, IEEE, Chengdu, China, May 2021.

[9] B. Zhang, Y. Yu, and J. Li, "Network intrusion detection based on stacked sparse autoencoder and binary tree ensemble method," in *Proceedings of the 2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–6, IEEE, Kansas City, MO, USA, May 2018.

[10] J. Xiao, L. Chun, J. Zhao, W. Jinxia, A. Hu, and G. Du, "A survey on network intrusion detection based on deep learning," *Frontiers of Data and Domputing*, vol. 3, no. 3, pp. 59–74, 2021.

[11] S. Zheng, "Network intrusion detection model based on convolutional neural network," in *Proceedings of the 2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pp. 634–637, IEEE, Chongqing, China, March 2021.

[12] S. Tharewal, M. W. Ashfaque, S. S. Banu, P. Uma, S. M. Hassen, and M. Shabaz, "Intrusion detection system for industrial Internet of Things based on deep reinforcement learning," *Wireless Communications and Mobile Computing*, vol. 2022, Article ID 9023719, 8 pages, 2022.

[13] P. Koopman, "Embedded system security," *Computer*, vol. 37, no. 7, pp. 95–97, 2004.

[14] A. Yan, Z. Chen, H. Zhang et al., "Effective detection of mobile malware behavior based on explainable deep neural network," *Neurocomputing*, vol. 453, pp. 482–492, 2021.

[15] P. Wang, X. Kong, G. Peng, X. Li, and Z. Wang, "Network intrusion detection based on deep learning," in *Proceedings of the 2019 International Conference on Communications Information System and Computer Engineering (CISCE)*, pp. 431–435, IEEE, Haikou, China, July 2019.

[16] K. Pragma, S. Banerjee, K. C. Mondal, G. Mahapatra, and S. Chattopadhyay, "A hybrid intrusion detection system for hierarchical filtration of anomalies," in *Information and Communication Technology for Intelligent Systems*, Springer, Berlin, Germany, 2019

[17] R. M. Elbasiony, E. A. Sallam, T. E. Eltobely, and M. M. Fahmy, "A hybrid network intrusion detection framework based on random forests and weighted k-means," *Ain Shams Engineering Journal*, vol. 4, no. 4, pp. 753–762, 2013.

[18] A. K. Balyan, S. Ahuja, U. K. Lilhore et al., "A hybrid intrusion detection model using EGA-PSO and improved random forest method," *Sensors*, vol. 22, no. 16, p. 5986, 2022.

[19] M. K. Yoon, S. Mohan, J. Choi, J.-E. Kim, and L. Sha, "SecureCore: a multicore-based intrusion detection architecture for real-time embedded systems," in *Proceedings of the 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 21–32, IEEE, Philadelphia, PA, USA, April 2013.

[20] E. Viegas, A. O. Santin, A. França, R. Jasinski, V. A. Pedroni, and L. S. Oliveira, "Towards an energy-efficient anomaly-based intrusion detection engine for embedded systems," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 163–177, 2017.

[21] T. Farid Molazem and K. Pattabiraman, "Flexible intrusion detection systems for memory-constrained embedded systems," in *Proceedings of the 2015 11th European Dependable Computing Conference (EDCC)*, pp. 1–12, IEEE, Paris, France, September 2015.

[22] A. Alhowaide and J. Alsmadi, "Ensemble detection model for IoT IDS," *Internet of Things*, vol. 16, no. 1, Article ID 100435, 2021.

[23] M. Almiani, A. AbuGhazleh, A. Al-Rahayfeh, S. Atiewi, and A. Razaque, "Deep recurrent neural network for IoT intrusion detection system," *Simulation Modelling Practice and Theory*, vol. 101, Article ID 102031, 2020.

[24] B. Zhang, "Research on weight calculation of Boltzmann machine based on hopfield network," *Computer & Telecommunication*, vol. 112, pp. 53–57, 2021.

[25] Y. Liu and X. Zhang, "Intrusion detection based on IDBM," in *Proceedings of the 2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, pp. 173–177, IEEE, Auckland, New Zealand, August 2016.

[26] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "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*, pp. 1–6, IEEE, Ottawa, Canada, September 2009.