Hindawi
Security and Communication Networks
Volume 2022, Article ID 4836289, 9 pages
https://doi.org/10.1155/2022/4836289

Research Article

Deep Neural Network-Based SQL Injection Detection Method

Wei Zhang[1], **Yueqin Li**[1], **Xiaofeng Li**[1], **Minggang Shao**[1], **Yajie Mi**[1], **Hongli Zhang**[1] **and Guoqing Zhi**[2]

[1]*Smart City College, Beijing Union University, Beijing 100101, China*
[2]*College of Applied Arts and Sciences, Beijing Union University, Beijing100191, China*

Correspondence should be addressed to Wei Zhang; 20201081210207@buu.edu.cn and Yueqin Li; xxtyueqin@buu.edu.cn

Received 31 December 2021; Accepted 2 March 2022; Published 24 March 2022

Academic Editor: Thi-Thu-Huong Le

Among the network security problems, SQL injection is a common and challenging network attack means, which can cause inestimable loop-breaking and loss to the database, and how to detect SQL injection statements is one of the current research hotspots. Based on the data characteristics of SQL statements, a deep neural network-based SQL injection detection model and algorithm are built. The core method is to convert the data into word vector form by word pause method, then form a sparse matrix and pass it into the model for training, build a multihidden layer deep neural network model containing ReLU function, optimize the traditional loss function, and introduce Dropout method to improve the generalization ability of this model. The accuracy of the final model is maintained at over 96%. By comparing the experimental results with traditional machine learning algorithms and LSTM algorithms, the proposed algorithm effectively solves the problems of overfitting in machine learning and the need for manual screening to extract features, which greatly improves the accuracy of SQL injection detection.

1. Introduction

The introduction should be succinct, with no subheadings. Limited figures may be included only if they are truly introductory and contain no new results.

Among the network security issues, SQL injection is a common and challenging means of network attacks, which is listed as the top 10 web application security risks by the Open Web Application Security Project (OWASP) and also listed as one of the top 10 vulnerabilities by OWASP in the past 15 years [1, 2] and network attacks caused by SQL injection. On average, web attacks caused by SQL injection cause nearly $10 billion loss to the US economy every year; therefore, how to effectively detect SQL injection is one of the current research hotspots.

However, most of the solutions proposed so far can detect only a subset of SQL injection attacks and cannot meet the challenge of changing attack methods. Therefore, for various types of SQL injection attacks and their countless variants, it is of considerable importance to study and design an effective deep learning-based detection scheme that can automatically perform feature extraction.

In recent years, researchers have also proposed many detection methods, which are mainly classified into three approaches: traditional-based, machine-learning-based, and deep neural network-based. The traditional-based approach scales well in traditional string matching, but nowadays, attackers use increasingly sophisticated tools for automatic injection attacks, and the traditional-based approach often cannot cope well. The machine learning-based approach solves the problems of the traditional approach but suffers from overfitting and the need for manual filtering to extract features [3]. Deep neural network-based approaches do not require manual feature extraction and do not suffer from overfitting.

The purpose of this study is to build a classification model for SQL injection detection using SQLNN deep neural networks. By analyzing a large amount of SQL injection data, relevant features are extracted, and then the neural network model is trained using a large amount of actual data. Finally, the model training results are compared.

LSTM, KNN, and DT algorithms are used for experimental model comparison, and the experimental results show that the detection effect of the deep neural network model proposed in this article is better.

2. Related Works

2.1. SQL Injection and Its Detection Method

2.1.1. SQL Injection Methods. SQL injection is an attacker who exploits a database vulnerability to change SQL statements so that the changed SQL statements can bypass the database security and enter the database and can perform a series of operations such as adding, deleting, modifying, and viewing the database [3]. That is, the attacker exploits the vulnerability by providing specially prepared input data so that the SQL interpreter cannot distinguish between the request command of the query and the attacker's specially prepared data [4]. Since the SQL language is rich and diverse and contains different encoding methods, there is a high probability of being attacked at any point in the dynamic construction of SQL statements. The developer often does not filter the user input sufficiently, and when the attacker splices the carefully constructed URL parameters or form submission parameters to the predefined SQL query, the SQL structure expected by the developer can be changed, and the execution of the SQL statement yields results that are no longer expected by the developer, forming a SQL injection attack [5].

2.1.2. SQL Injection Detection Method

(1) Traditional Method. Traditional SQL injection mostly uses filtering allergy characters to prevent SQL injection. In "SQL Injection Attack and Defense", whitelist validation and blacklist validation are two different types of input validation methods to defend against SQL injection [6]. Penetration testing can make up for the deficiencies of blacklist and whitelist filtering defense mechanisms, but it cannot fundamentally solve the deficiencies [7]. Halfond and Orso [8] developed a tool, AMNESIA, based on traditional blacklisting techniques, implementing a combined dynamic and static approach, which in its static part automatically constructs a model of legitimate queries that can be generated by the application. In its dynamic part, it examines dynamically generated queries and checks them against the statically constructed model.Xiao et al. proposed a method to detect SQL injection based on URL-SQL mapping to analyze user behavior and execution responses [9]. This method does not incur much additional cost for web applications, but the system execution has an uncertainty factor and the extraction of invariants (normal state of the web application) is not comprehensive, leading to the generation of a large number of false alarms and missed alarms [5].

These methods above can scale well with the traditional string matching at that time, but nowadays, attackers use increasingly sophisticated tools for automatic injection attacks and traditional methods often do not cope well [5].

(2) *Machine Learning.* The application of machine learning in SQL injection is already a common phenomenon. Joshi et al. [10] designed a classifier that consists of a Naïve Bayes machine learning algorithm and a role-based access control mechanism for detecting SQL injection attacks and tested the model with three SQLIA attacks: annotation, union, and restatement. Kamtuo and Soomlek [11] proposed a decision tree-based SQL injection prevention framework and used 1100 vulnerability datasets to train the machine learning model. Wu and Chen [12] proposed a method called k-centers (KC) to detect SQL injection attacks, which is based on k - means clustering algorithm for hybrid data [13], and traditional machine learning algorithms by adapting the number and location of different types of attacks in KC clusters. However, machine learning-based methods [3, 14–16] have their own limitations, such as the inability to detect attacks using injection evasion techniques [17], overfitting problems [18], and the need for manual filtering to extract features. In addition, most of these methods are based on raw query string analysis and cannot take advantage of the latest machine learning techniques and the contextual and syntactic information of the available SQL strings [19].

(3) Deep Neural Networks. Deep neural networks, also called deep learning (DL, Deep Learning), is a new research direction in the field of machine learning (ML, Machine Learning), which was introduced into machine learning to bring it closer to its original goal, artificial intelligence (AI, Artificial Intelligence) [20]. Deep learning is the process of learning the intrinsic laws and levels of representation of sample data, and the information obtained from these learning processes can be of great help in the interpretation of data such as text, images, and sounds [20]. Its ultimate goal is to enable machines to have analytical learning capabilities like humans, capable of recognizing data such as text, images, and sounds. Deep learning is a complex machine learning algorithm that has achieved results in speech and image recognition that far exceed previous related techniques [20]. Deep learning has also been successively applied to web security detection in recent years. Sirinam et al. [21] used fingerprinting attacks on a CNN defense website and showed that the accuracy of CNN for fingerprinting attack detection on this website was above 98%. Yuan et al. [22] provided a subspace spectrum integration clustering method, called depe - ssec, that supports deep learning for web attack detection. Selvaganapathy et al. [23] used deep confidence networks to extract URL features and used deep neural networks to classify normal and malicious URLs.

3. Characterization

SQL injection detection is essentially a classification problem. In SQL injection detection, data are divided into two main categories: SQL injection statements and non-SQL injection statements. The non-SQL injection statements are further divided into the following: ordinary text statements and ordinary SQL statements. The data set is marked into two categories: SQL injection statements are marked as 1 and non-SQL injection statements are marked as 0. The main purpose of SQL injection detection is to identify the aggressive SQL injection statements and the nonaggressive non-SQL injection statements: ordinary SQL statements and ordinary text statements can be grouped into one category. Therefore, the model can distinguish the aggressive SQL statements from the nonaggressive ones. Based on the research in this article, the dataset used in this article has a total of 30919 items, which are divided into the following two categories.

3.1. SQL Injection Statements. During the current study, SQL injection statements typically contain the following keywords:, ∗ ,; , _, -, (), = , {, }, @,.,, and, [,], +, −, ?, %, !,:, \,/. Also, SQL injection statements contain this common SQL token, for example, where, table, like, select, update, and, or, set, like, in, having, values, into, alter, as, create, revoke, deny, convert, exec, concat, char, tuncat, ASCII, any, asc, desc, check, group by, order by, delete from, insert into, drop table, union, join. Based on the dataset used in this article, the SQL injection statements are classified into the following types:

(1) Repetition attack: the main purpose is to detect whether there is an injection point by the truth or falsity of the expression. The SQL language contains keywords such as where and group by, which are generally followed by query conditions. The attacker inserts the true-true formula into the position of the conditional statement so that the query condition is always identified as true, which can bypass the page authentication and obtain sensitive data.

(2) Illegal comment attack: the SQL language contains various forms of comments, such as --, #,/ ∗ /, whose subsequent statements will not be executed. The attacker often inserts these comment characters into the conditional SQL statement so that the legitimate SQL statement after the comment character cannot be executed to achieve the injection purpose.

(3) Union query attack: in SQL language, keywords such as join, left join, and union are used for the union implementation of one or more statements. An attacker can use the union keyword to add malicious SQL code after the legitimate code to obtain additional sensitive information by controlling the second statement to perform illegal operations.

(4) Explicit error injection: the attacker makes the application display default error pages by executing illegal or logically incorrect SQL queries. These pages often reveal to the attacker information such as injectable parameters and the type of database.

(5) Boolean blind injection: the main purpose is to submit queries with a different logic and observe whether the page returned by the web application is normal to determine whether there is an injection. Boolean blinding is often used to obtain information such as the name of the database.

(6) Time blind injection: it mainly refers to inserting a time delay function in the URL or user input to determine whether the injection is successful by observing whether there is a delay in the web application's response.

(7) Multistatement attack: this attack generally uses query separators to add additional queries to the original query to extract, add, and modify data or execute remote commands. The DBMS receives multiple SQL queries, the first is the normal execution of the query, and the subsequent queries are executed to meet the attack.

SQL injection statements using the above keywords and injection types can bypass the security defenses of the database and enter the database with a false identity. After the database is invaded, it does not detect such enemies by itself and only allows such enemies to wreak havoc in the database, such as adding, deleting, modifying, and viewing tables.

3.2. Non-SQL Injection Statements

(1) *Plain Text Statements*. Ordinary text statement is composed of ordinary letters, numbers, and characters. The statement is not offensive in any way and will be recognized directly in the detection as a nonformal statement.

(2) *General SQL Statements*. General SQL statements are the SQL statements that users encounter every day and are used in the daily maintenance of the database, for example, creating, querying, and modifying tables. These types of SQL statements usually contain keywords such as rename, drop, delete, insert, create, exec, update, union, set, alter, database, and, or, information_schema, load_file, select, shutdown, cmdshell, hex, ascii,etc. It will also contain some dangerous keywords: --, #,/ ∗ , ‘, ”, ||, \\, = .

Example data are shown in Table 1.

3.3. Data Characterization. Analysis of the data shows that words such as “select”,“ ∗ ”,“from” appear very frequently in SQL injection statements and non-SQL injection statements, and these very frequent words do not serve as features for classification. Therefore, in this article, when using the TF-IDF algorithm to process the data, such words can be filtered out sufficiently to achieve dimensionality reduction.

4. Model Design

Based on the data characteristics of SQL statements, a deep neural network-based SQL injection detection model-SQLNN is built, which can effectively detect SQL injection statements. The model includes data processing, model training, and model evaluation when performing SQL injection detection, as shown in Figure 1.

Table 1: SQL injection detection sample data.

Description	Label
select ∗ from users where id = 1 or " ( ]" or 1 = 1 -- 1	1
select ∗ from users where id = 1 or ".@" or 1 = 1 -- 1	1
15495714q	0
4ibari8a	0
Select ∗ from general where easy between 10 and 20	0
Select ∗ from case where century not between "silk" and "guess"	0

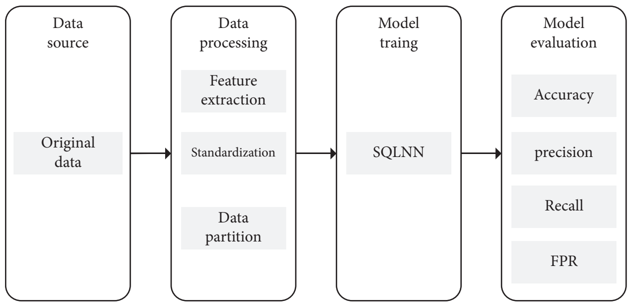

Figure 1: Schematic diagram of SQL injection detection training process.

4.1. IF-IDF. When training classifiers using deep neural networks, SQL injection samples need to be digitized before being fed into the model because the computer cannot understand them directly. SQL injection samples, like natural language, are text with specific grammatical rules. There are several text vectorization methods in natural language processing, such as word set models, BoW, TF-IDF, and distributed word vectors. TF- IDF algorithm can measure the importance of words by word frequency and inverse document rate. Therefore, the TF-IDF algorithm is used to process the sample data in this article.

The data processing layer preprocesses the metadata, including feature extraction, normalization, and data segmentation. In the data processing, the model first tokenizes the data using the word pause method, a token represents a word, and the TF-IDF algorithm is used to filter out the common words and keep the important words. A token with an 80% occurrence rate and a token with less than 2 occurrences in the dataset indicates that the word occurs equally frequently in both SQL injection and non-SQL injection statements and cannot be used as a feature to distinguish between SQL injection statements and non-SQL injection statements. A frequency of fewer than 2 occurrences indicates that the occurrence rate of the term in both SQL and non-SQL statements is extremely low, and it cannot be used as a feature to distinguish whether it is a SQL injection statement or not.

TF, or word frequency, indicates how often a given word appears in a piece of text, as in the following formula:

$$TF = \frac{\text{Number of times a word appears in the text}}{\text{Total number of words in the text}}. \tag{1}$$

Word frequency alone cannot accurately describe the importance of a word in a text because some words may repeatedly appear in many texts, such as "The" and "An" in English and "I" and "you" in Chinese. IDF is the inverse text frequency, which reflects the frequency of a given word in all texts. IDF value will be lower if the word occurs in many texts, as in the following formula:

$$IDFx = \log\frac{N+1}{Nx+1} + 1, \tag{2}$$

where N denotes the total number of texts in the training set and $N(x)$ denotes the number of texts containing word x.

The final result of the TF-IDF algorithm is the result of multiplying the two values of word frequency and inverse text frequency, which indicates the importance of a word in the text, as in the following formula:

$$TF - IDFx = TF \times IDFx. \tag{3}$$

Using IF-IDF algorithm processing, some words that cannot be used as feature values are filtered out, thus achieving data dimensionality reduction. After the above processing, the extracted token words are vectorized and then converted into the form of a sparse matrix as the input data for subsequent model training.

4.2. Data Preprocessing. SQL injection attack statements have characteristics that are different from those of normal URLs. Not only do these attacks contain SQL-related keywords, but the form of the parameters is more complex than normal parameters. How to extract the main keywords from SQL injection statements as detection features is a very important step in data processing, and the extracted keywords must be able to distinguish whether the statement is offensive or not, and the extracted keywords must also be able to express the original meaning of the sentence. Traditional machine learning methods require manual screening of the feature values, which is laborious and difficult, and cannot select the most representative ones. Therefore, this model filters out most of the words in the utterance in the form of deactivated words using the TF-IDF algorithm and then normalizes them. The keywords left in the utterance are encoded numerically, and the whole data can be represented by a sparse matrix of word vectors: $D = \begin{array}{c} q_1 \ \cdots \ q_k \\ \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix} \end{array}$. $\mathbf{q}_1 \ldots \mathbf{q}_\mathbf{k}$ denotes the k keywords (k features) extracted, and the presence of this feature in the sample denotes the feature as 1 (greater than 1 denotes the number of occurrences); otherwise, it denotes 0. The specific algorithm steps are as follows:

(1) Input: dataset $X = \{X_1, X_2, \cdots, X_n\}$, tag $y = \{y_1, y_2, \cdots, y_n\}\}$.

(2) Segment the dataset with word pauses, tokenize each data, one token represents one word, and each data is represented as multiple tokens.

(3) Filter out tokens with an 80% occurrence rate and tokens with less than 2 occurrences. The 80% occurrence rate means that the word appears quite

frequently in both SQL injection and non-SQL injection statements, so it cannot be used as a feature value to distinguish whether it is a SQL injection statement or not. If the occurrence rate is less than 2 times, the occurrence rate of the term in both SQL and non-SQL statements is extremely low, and it cannot be used as a feature to distinguish whether it is a SQL injection statement or not.

(4) Vectorize the filtered token to form a vector matrix.

(5) Transform the vector matrix into a sparse matrix, denoted as $D = \begin{array}{c} q_1 \cdots \mathbf{q_k} \\ \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix} \end{array} s$. $\mathbf{q}_{1\ldots}\mathbf{q_k}$ denotes k keywords extracted (k features), and the presence of this feature in the sample is represented as 1 (greater than 1 indicates the number of occurrences); otherwise, it is represented as 0.

It can be seen that when using this model for SQL injection detection, the feature values that facilitate the distinction between SQL injection statements and non-SQL injection statements can be effectively selected, thus achieving data dimensionality reduction and solving the difficulty of manually extracting feature values in machine learning. This model introduces word vectors into SQL injection detection, which facilitates the calculation in neural network training and is conducive to improving the performance analysis index at a later stage.

4.3. Deep Neural Network Model-SQLNN Model. The SQLNN deep neural network constructed in this article includes an input layer, an output layer, and three hidden layers, where both the hidden layer and the output layer have processing capabilities. The deep neural network with a single hidden layer can be regarded as a special kind of logistic regression classifier, which first performs a nonlinear transformation on the input data and then uses the result as the input for logistic regression. The nonlinear transformation maps the input samples to a linearly separable space. In this model, the input layer reduces the incoming data to 64 dimensions, and it consists of multiple input nodes that pass the transformed information to the hidden nodes. The hidden layer consists of multiple hidden nodes with no external connections. The output of the hidden nodes depends on the output of the input layer and the weights attached to the connections and passes the result to the next hidden layer or output layer. The output layer consists of multiple output nodes that take data from the hidden layer and perform similar calculations as the hidden layer. The final result is the output of this model.

4.3.1. Nonlinear Transformation of Data in Forward Propagation. The input data of the model starts from the input layer and is computed layer by layer and propagated backward sequentially through the network parameters and activation functions. In the whole model, the output of the previous layer of the network is used as the input of the next layer of the network up to the output layer of the model. In SQL injection detection, since the sample data are not all linearly separable, problems such as data scattering and network gradient disappearance will occur during the propagation process, so it is necessary to introduce an activation function in the implicit layer to transform the data nonlinearly during the specific implementation of both sides. Here, the Rectified Linear Unit (ReLU) function [24] is applied to complete the nonlinear transformation of the data, and the ReLU function does not activate all neurons, making the neural network efficient and easy to compute.

As shown in Figure 2, the input layer data in the model is the feature data of $X = [x_1, x_2, \cdots, x_N]^T$ processed by the IF-IDF algorithm, and its data dimension is N. The activation functions of the three hidden layers are chosen as ReLU, which is denoted as $f(x) = \max(0, x)$. The output vectors of the three hidden layers are denoted as H_1, H_2, and H_3, respectively, and the data dimensions are, respectively, n_1, n_2, $n_3 n_3$:

$$\begin{aligned} H_1 &= f(W_1 X + b_1), \\ H_2 &= f(W_2 H_1 + b_2), \\ H_3 &= f(W_3 H_2 + b_3). \end{aligned} \tag{4}$$

W_1, W_2, and W_3 denote the weights of the three hidden layers, respectively. The weights are the connections between neurons, which calculate an output from the data input and then compare it with the actual output, the error is back-propagated, and the weights are continuously adjusted, which in turn reduces the error.

The sigmoid function selected for the output layer [25], the output of the sigmoid function is between 0 and 1. The output of the output layer of this model is the probability of the classified event, which is classified when certain probability conditions are satisfied. So the sigmoid function is suitable as the output layer of this model.

4.3.2. Optimization of Loss Function. The output error of the model is obtained by comparing the model output prediction with the target true category. In training the model, we use Cross-Entropy [26] (Cross-Entropy Loss) as the loss function to describe this error between the model predicted value and the known true value, noted as follows:

$$L = -\frac{1}{N} \sum_{i=1}^{N} [y_i \cdot log(p_i) + (1 - y_i) \cdot log(1 - p_i)], \tag{5}$$

where N is the number of training samples and $y = [y_1, y_2 \ldots y_n]^T$; y_i represents the expected output value of the ith sample, i.e., the true label of the sample. p_i denotes the probability that the ith sample is predicted to be a positive case.

When performing SQL injection detection, the loss function is continuously reduced to achieve a higher recognition rate during training. However, the reduction of the loss function using the traditional Stochastic Gradient Descent (SGD) [27] makes the optimized DNN converge slowly and easily fall into local optimal solutions. For this reason,

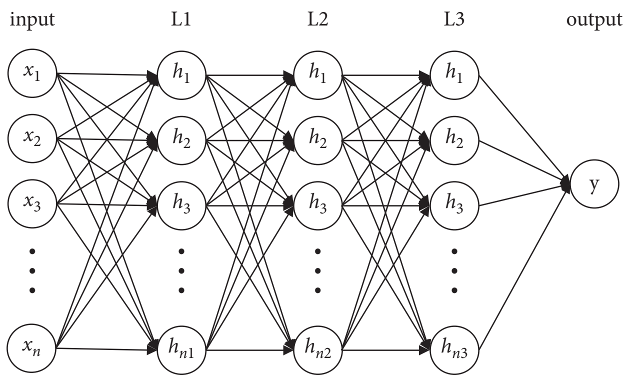

Figure 2: Schematic diagram of the neural network training process.

the Adaptive Moment Estimation (Adam) algorithm [28] is used instead of the traditional SGD method to update the model parameters. The obtained model training errors are used to calculate the updated values of the weights and bias vectors using the backpropagation algorithm, and the weight parameters and bias vectors of the network model are readjusted according to the obtained results. Assuming that the small batch random gradient of parameter δ in the training round is g_t, m_t is the first-order moment estimate of the gradient, v_t is the second-order moment estimate of the gradient, and β_1, β_2 corresponds to the exponential decay rate of m_t, v_t; then,

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t, \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2. \end{aligned} \tag{6}$$

Then the deviation correction is made for m_t and v_t, which are written as follows:

$$\begin{aligned} \widehat{m}_t &= \frac{m_t}{1-\beta_1^t}, \\ \widehat{v}_t &= \frac{v_t}{1-\beta_2^t}. \end{aligned} \tag{7}$$

The final parameter is updated to $\theta_{t+1} = \theta_t - \eta \cdot \widehat{M}_t / (\sqrt{\widehat{v}} + \varepsilon)$, where η is the learning rate and ε is the smoothing term.

Adam's algorithm can iteratively update the neural network weights based on SQL injection training samples for more effective learning and can correct the problems of disappearing learning rate and large fluctuations in the loss function during the training process [28]. Thus, it achieves higher recognition rates by minimizing the loss function to the maximum extent while accelerating convergence and learning correctly.

4.3.3. Using Dropout [29] to Prevent Overfitting. For deep neural network models, the training difficulty of the network increases gradually as the depth of the network increases, and the overfitting problem is prone to occur when the training samples of SQL injection detection features are small [16]. Therefore, the Dropout method is introduced in the training process of the proposed deep neural network in this article, which improves the generalization ability of the model while alleviating the model overfitting problem. In this article, the neural network contains three hidden layers, and after introducing the Dropout method, some neurons will be discarded with a certain probability during the training process of the neural network by setting different discard probabilities. Through this operation, the generalization performance of the model is effectively improved after enough iterations of the neural network, and the risk of the model falling into a state of overfitting during the training process is effectively reduced [29]. In the testing phase, Dropout restores the connections between all neurons to ensure that the best recognition performance is obtained when the model is tested.

5. Results and Discussion

In order to verify the effectiveness of the proposed algorithm, the results of SQL injection detection by traditional machine learning algorithms (KNN algorithm and DT algorithm) and LSTM algorithm are compared and analyzed with the deep neural network model algorithm proposed in this paper. The algorithms developed in this study were developed using the *Python* programming language using the Keras and TensorFlow libraries. The hardware platform is a computer with an i7 9900 HQ CPU and 16 GB of RAM.

5.1. Datasets. The dataset in this article is from the dataset published on https://www.kaggle.com/sajid576/sql-injection-dataset, which has a total of 30,919 data items and basically meets the experimental requirements. The training set is 70% of the dataset and the test set is 30%. The dataset is shown in Table 2.

5.2. Evaluation Indicators. During the training of a deep learning model, we can get multiple classifiers, and we need to evaluate the performance of each classifier using appropriate evaluation metrics, from which the best one is selected. The samples can be combined according to the real target category and the category predicted by the classification model to obtain the following four cases: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). Let TP, TN, FN, and FP denote their corresponding sample numbers, then obviously TP + TN + FN + FP = total number of samples. The confusion matrix of the classification results by taking each class as positive samples separately is shown in Table 3.

For classification models, the evaluation criteria are Accuracy, Recall, ROC, etc. Since positive and negative sample imbalance is very common in the field of SQL injection attack detection, it is unreasonable to use only accuracy rate as the evaluation metric, so the evaluation metric used is F1-Score as the detection classifier performance in addition to detection accuracy (Accuracy), check-all rate (Recall), and check-accuracy rate (Precision). The F1-Score is used as a comprehensive evaluation criterion for classifier performance.

Table 2: Data situation of SQL injection detection.

Label	Description	Count	Ratio (%)
1	SQL injection statement	11330	36.64
0	Non-SQL injection statement	19589	63.36

Table 3: Confusion matrix schematic table.

True	Forecast	
	Positive	Counter
Positive	TP	FN
Counter	FP	TN

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FN+FP},$$
$$\text{Precision} = \frac{TP}{TP+FP},$$
$$\text{Recall} = \frac{TP}{TP+FN}, \tag{8}$$
$$F1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}.$$

5.3. Comparative Analysis. The larger the percentage of TP and TN values in the confusion matrix, the higher the Accuracy, Recall, Precision, and F1 indexes of the model, the better the classification effect of the model. The KNN, DT, and LSTM methods are compared with the present model for the test. The confusion matrix of the four methods is obtained as shown in Tables 4–7.

The comparative analysis of SQL injection attack studies is shown in Table 8, where we compare LSTM, KNN, DT, and our own proposed model using four metrics, Accuracy, Precision, Recall, and F1. The model proposed in this article outperforms other existing models in these four metrics and has a lower error rate, indicating that the method proposed in this article can effectively detect SQL injection statements. The approach in this article focuses on as many features as possible and uses deep neural networks to make the proposed model more robust, so it can effectively detect all types of SQL injection attack queries. The comparison results are shown in Table 8.

KNN and DT algorithms suffer from the inability to detect attacks using injection evasion techniques [17], overfitting problems [18], and the need for manual filtering to extract features. In addition, most of these methods are based on raw query string analysis and cannot take advantage of the latest machine learning techniques and the contextual and syntactic information of the available SQL strings [19]. LSTM, although solving the gradient problem of RNN to some extent, still seems tricky when dealing with long sequences. Also, LSTM is a time-consuming algorithm. The SQLNN model proposed in this article, which is based on a deep neural network, can effectively solve the problems of the above algorithms.

Table 4: Confusion matrix for SQLNN model.

SQLNN		
True	Forecast	
	Positive	Counter
Positive	TP (62.41%)	FN (0.53%)
Counter	FP (3.26%)	TN (33.80%)

Table 5: Confusion matrix for KNN model.

KNN		
True	Forecast	
	Positive	Counter
Positive	TP (49.88%)	FN (13.06%)
Counter	FP (4.23%)	TN (32.83%)

Table 6: Confusion matrix for DT model.

DT		
True	Forecast	
	Positive	Counter
Positive	TP (58.97%)	FN (3.97%)
Counter	FP (3.84%)	TN (33.22%)

Table 7: Confusion matrix for LSTM model.

LSTM		
True	Forecast	
	Positive	Counter
Positive	TP (37.39%)	FN (19.34%)
Counter	FP (23.95%)	TN (19.32%)

Table 8: Model comparison results.

Models	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
LSTM	62.32	66.23	65.16	64.23
KNN	82.69	71.51	88.56	79.13
DT	92.33	89.58	89.74	89.66
SQLNN	96.16	97.28	92.23	94.68

6. Conclusions

In this article, we propose a SQLNN deep neural network model. The core method is to convert the data into word vector form by word pause and then form a sparse matrix and pass it into the model for training to build a multi-hidden layer deep neural network model containing ReLU function, which optimizes the traditional loss function and introduces the Dropout method to improve the generalization ability of this model. In the comparison of KNN, DT, LSTM models, and the SQLNN model proposed in this article, the model in this article outperforms the other three models in four indexes, Accuracy, Precision, Recall, and F1, and effectively solves the problems of overfitting and the need for manual screening to extract features in machine learning. The accuracy rate of this model can be maintained above 96%, indicating that this model outperforms the other three models in SQL injection detection.

Data Availability

The dataset of this article was obtained from the dataset published on https://www.kaggle.com/sajid576/sql-injection-dataset, which has a total set of 30919 data, basically meeting the experimental requirements.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

References

[1] OWASP, https://owasp.org/www-project-top-ten/, 2020.

[2] U. Farooq, "Real time password strength analysis on a web application using multiple machine learning approaches," *International Journal of Engineering Research and Technology*, vol. 9, no. 12, pp. 359–364, 2020.

[3] P. R. McWhirter, K. Kifayat, Q. Shi, and B. Askwith, "SQL Injection Attack classification through the feature extraction of SQL query strings using a Gap-Weighted String Subsequence Kernel," *Journal of information security and applications*, vol. 40, pp. 199–216, 2018.

[4] Y. Vural and Ş. Sagiroglu, "A review on enterprise information security and standards," *Journal of the Faculty of Engineering and Architecture of Gazi University*, vol. 23, no. 2, 2008.

[5] F. Wang, *Research and Implement of SQL Injection Detection Technology Based on Deep Learning*, Beijing University of Posts and Telecommunications, Beijing, China, 2020, In Chinese, Article ID 000187.

[6] J. Clarke, *SQL Injection Attacks and defense*, Elsevier, Amsterdam, Netherlands, 2009.

[7] W. Tian, J. F. Yang, J. Xu, and G.-N. Si, "Attack Model Based Penetration Test for SQL Injection vulnerability," in *Proceedings of the 2012 IEEE 36th Annual Computer Software and Applications Conference Workshops. IEEE*, pp. 589–594, Izmir, Turkey, July 2012.

[8] W. G. J. Halfond and A. Orso, "AMNESIA: analysis and monitoring for neutralizing SQL-injection attacks," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pp. 174–183, California, CA, USA, November 2005.

[9] Z. Xiao, Z. Zhou, W. Yang, and C. Deng, "An Approach for SQL Injection Detection Based on Behavior and Response analysis," in *Proceedings of the 2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN)*, pp. 1437–1442, IEEE, Guangzhou, China, May 2017.

[10] A. Joshi and V. Geetha, "SQL Injection Detection Using Machine learning," in *Proceedings of the 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, pp. 1111–1115, IEEE, Kanyakumari, India, July 2014.

[11] K. Kamtuo and C. Soomlek, "Machine Learning for SQL Injection Prevention on Server-Side scripting," in *Proceedings of the 2016 International Computer Science and Engineering Conference (ICSEC)*, pp. 1–6, IEEE, Chiang Mai, Thailand, December 2016.

[12] X. R. Wu and P. P. K. Chan, "SQL injection attacks detection in adversarial environments by k-centers," *IEEE* in *Proceedings of the 2012 International Conference on Machine Learning and Cybernetics*, vol. 1, pp. 406–410, Xi'an, China, July 2012.

[13] W. D. Zhao, W. H. Dai, and C. B. Tang, "K-centers Algorithm for Clustering Mixed Type data," in *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 1140–1147, Springer, Berlin, Heidelberg, May 2007.

[14] R. Rawat and S. Kumar Shrivastav, "SQL injection attack Detection using SVM," *International Journal of Computers and Applications*, vol. 42, no. 13, pp. 1–4, 2012.

[15] A. Moosa, "Artificial neural network based web application firewall for SQL injection," *International Journal of Computer and Information Engineering*, vol. 4, no. 4, pp. 610–619, 2010.

[16] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[17] P. Yu and S. Wang, "Deep Tree: Sql Injection Detection by the Power of Deep Learning," 2018, https://github.com/Aetf/tensorflow-tbcnn/blob/master/misc/deeptree.pdf.

[18] Y. Fang, J. Peng, L. Liu, and C. Huang, "WOVSQLI: Detection of SQL Injection Behaviors Using Word Vector and LSTM," in *Proceedings of the 2nd International Conference on Cryptography, Security and Privacy*, pp. 170–174, Guiyang, China, March 2018.

[19] Z. Zhuo, T. Cai, X. Zhang, and F. Lv, "Long short-term memory on abstract syntax tree for SQL injection detection," *IET Software*, vol. 15, no. 2, pp. 188–197, 2021.

[20] X. C. Chen, *Research on Algorithm and Application of Deep Learning Based on Convolutional Neural Network*, Zhejiang University of Commerce and Industry, Zhejiang, China, 2014, In Chinese.

[21] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1928–1943, Toronto Canada, October 2018.

[22] G. Yuan, B. Li, Y. Yao, and S. Zhang, "A Deep Learning Enabled Subspace Spectral Ensemble Clustering Approach for Web Anomaly detection," in *Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 3896–3903, IEEE, Anchorage, AK, USA, May 2017.

[23] S. Selvaganapathy, M. Nivaashini, and H. Natarajan, "Deep belief network based detection and categorization of malicious URLs," *Information Security Journal: A Global Perspective*, vol. 27, no. 3, pp. 145–161, 2018.

[24] A. F. Agarap, "Deep Learning Using Rectified Linear Units (relu," 2018, https://arxiv.org/abs/1803.08375.

[25] J. Han and C. Moraga, "The Influence of the Sigmoid Function Parameters on the Speed of Backpropagation learning," in *Proceedings of the International Workshop on Artificial Neural Networks*, pp. 195–201, Springer, Berlin, Heidelberg, June 1995.

[26] Z. Zhang and M. R. Sabuncu, "Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy labels," in *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[27] N. Ketkar, "Stochastic gradient descent," Apress, Berkeley, CA, pp. 113–132, 2017.

[28] E. Okewu, S. Misra, and F.-S. Lius, “Parameter Tuning Using Adaptive Moment Estimation in Deep Learning Neural networks,” in *Proceedings of the International Conference on Computational Science and its Applications*, pp. 261–272, Cagliari, Italy, Julu 2020.
[29] Y. Gal, J. Hron, and A. Kendall, “Concrete dropout,” 2017, https://arxiv.org/abs/1705.07832.