Journal of Computer and Communications, 2021, 9, 1-21
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

# Application of Weighted Cross-Entropy Loss Function in Intrusion Detection

**Ziyun Zhou[1], Hong Huang[1,2*], Binhao Fang[1]**

[1]School of Computer Science and Engineering, Sichuan University of Science and Engineering, Yibing, China
[2]Key Laboratory of Higher Education of Sichuan Province for Enterprise Informationalization and Internet of Things, Yibing, China
Email: *huanghong@suse.edu.cn

**How to cite this paper:** Zhou, Z.Y., Huang, H. and Fang, B.H. (2021) Application of Weighted Cross-Entropy Loss Function in Intrusion Detection. *Journal of Computer and Communications*, **9**, 1-21.
https://doi.org/10.4236/jcc.2021.911001

**Received:** September 29, 2021
**Accepted:** November 5, 2021
**Published:** November 8, 2021

## Abstract

The deep learning model is overfitted and the accuracy of the test set is reduced when the deep learning model is trained in the network intrusion detection parameters, due to the traditional loss function convergence problem. Firstly, we utilize a network model architecture combining Gelu activation function and deep neural network; Secondly, the cross-entropy loss function is improved to a weighted cross entropy loss function, and at last it is applied to intrusion detection to improve the accuracy of intrusion detection. In order to compare the effect of the experiment, the KDDcup99 data set, which is commonly used in intrusion detection, is selected as the experimental data and use accuracy, precision, recall and F1-score as evaluation parameters. The experimental results show that the model using the weighted cross-entropy loss function combined with the Gelu activation function under the deep neural network architecture improves the evaluation parameters by about 2% compared with the ordinary cross-entropy loss function model. Experiments prove that the weighted cross-entropy loss function can enhance the model's ability to discriminate samples.

## Keywords

Cross-Entropy Loss Function, Visualization Analysis, Intrusion Detection, KDD Data Set, Accuracy

## 1. Introduction

Intrusion detection system can be regarded as a kind of active defense of computer network, and it was created to ensure the security of information communication. At the moment, affected by the 2020 epidemic, most people's live and work are almost closely related to the Internet, and the amount of data has also increased dramatically, and at the same time, we are facing data abuse, data security issues such as attacks and theft have also surged. These security issues make us face many challenges; this also makes us pay more attention to intrusion detection systems.

First, machine learning was first applied to intrusion detection because it is a fairly intelligent technology that automatically obtains knowledge from massive datasets [1] [2] [3]. With machine learning IDS, IDS can be better detected if enough training data is available for learning. ML is largely independent of knowledge in related fields, which makes it easier to build models.

Nowadays, machine learning methods have been widely used in various types of network intrusion detection, and there are many analysis methods based on machine learning, such as KNN, SVM, decision tree, Bayesian algorithm and so on. With the rapid development of network equipment and related technologies, massive amounts of network data have been generated. Traditional machine learning algorithms have become increasingly difficult to solve the classification problem of massive intrusion data in actual networks. Deep learning is a new research direction in the field of machine learning [4]. Its network model contains multiple hidden layers of multi-layer perception institutions. By combining the underlying features to form a more abstract high-level representation attribute category or feature, it can discover the distributed characteristics of the data.

## 2. Related Works

At present, applying deep learning technology to the design of intrusion detection systems can effectively improve the accuracy and efficiency of intrusion detection. Andresini *et al.* [5] proposed a novel deep learning method that uses a convolutional neural network (CNN) to equip a computer network with an effective means to analyze the traffic on the network to find signs of malicious activity. The basic idea is to represent the network stream as a 2D image and use the image representation of this stream to train the 2D CNN architecture. But the training effect is not stable. Michał *et al.* [6] compared a wide range of ANN settings, conducted experiments on two benchmark data sets and improved the accuracy of multi-classification. However, if the training parameters are not selected properly, the F1-Score will be low. Mighan *et al.* [7] used Apache Spark as a big data processing tool to process a large amount of network traffic data. In addition, they proposed a hybrid scheme that combines the advantages of deep network and machine learning methods to improve the accuracy of detection. But the disadvantage is that the network structure is simple and the accuracy is low. The tree-CNN-based classifier algorithm proposed by Mendonça *et al.* [8], Improving the efficiency of detection. Andresini *et al.* [9] proposed a new intrusion detection method, this method analyzes the flow-based characteristics of network traffic data and it learns the intrusion detection model by using the deep metric learning method that originally combined the autoencoder and the triplet network. Khan *et al.* [10] used Convolutional Recurrent Neural Network (CRNN) to create a DL-based hybrid ID framework that can predict and classify malicious network attacks in the network. In HCRNNIDS, Convolutional Neural Network (CNN) performs convolution to capture local features, and Recurrent Neural Network (RNN) captures temporal features to improve the performance and prediction of the ID system. Sajith *et al.* [11] used computational intelligence algorithms such as genetic algorithm (GA), genetic programming (GP) and swarm intelligence algorithm to determine the optimization of interesting rules from dense databases.

Among these examples of integrating deep learning into intrusion detection systems, in fact, there are many examples that often lead to different convergence speeds due to different selection of loss functions, which affects the model training over-fitting and reduces the training accuracy instead. The DNN + Gelu algorithm uses Relu and Gelu activation functions in each layer of its neural network to work together to extract different data features to improve the generalization ability and accuracy of the algorithm. The weighted Cross-Entropy loss function is used to solve the problem that the accuracy of the deep learning model overfitting on the test set due to the imbalance of the convergence speed of the loss function decreases.

## 3. Deep Neural Network Model

Deep Neural Network (DNN) can be understood as a neural network with many hidden layers, also known as Deep Feed Forward Network (DFN), Multi-Layer Perceptron (MLP), First divide the DNN according to the position of different layers, the internal neural network of DNN can be divided into three layers, input layer, hidden layer and output layer. In general, the first layer is the input layer, the last layer is the output layer, and the middle part is the hidden layer. [12] Then the DNN deep neural network is not only layered but also divided into transmission directions, which are forward and backward respectively, the forward tim9e data passes through n hidden layers from the input layer after preprocessing, and passes it to the output layer after calculation, and then compares the output result after the output layer is activated with the expected result. After comparison, the error is found, and then the error is passed from the output layer through the hidden layer back to the input layer in a gradient descent manner, which completes a round of neural network training [13]. The structure diagram of the deep neural network is shown in **Figure 1**.

### 3.1. Fully Connected Layer

The fully connected layer uses the form of a cooperative activation function, and divides the output of each layer into two parts on average, and uses the Relu activation function and the Gelu activation function to perform non-linear classification respectively.

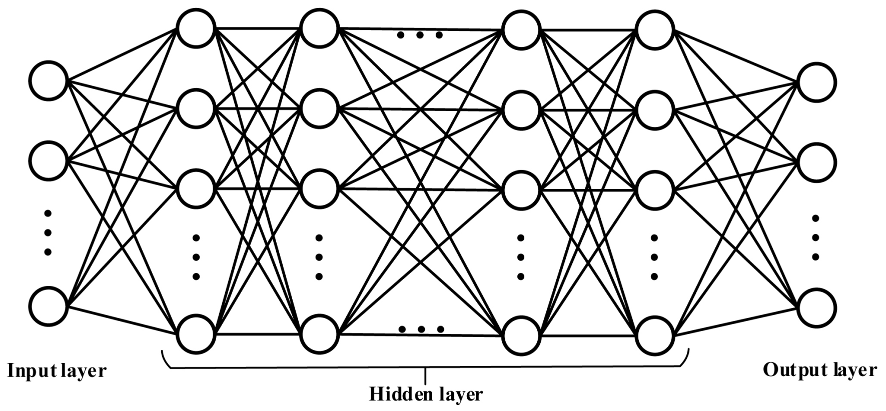

Figure 1. Deep neural network structure diagram.

In the neuron, after the input layer is weighted and summed, a function is also applied. This function is the activation function. The activation function is a very important part of the neural network. It can perform a nonlinear transformation on the information received by the neuron and output the transformed information to the next layer of neurons. If the activation function is not used, then the output of each layer is the linear function of the previous layer, no matter how many layers there are in the neural network, the output is the linear combination of the previous layer [14]. After using the activation function, we can introduce non-linear factors to the neuron. At this time, the neural network can approximate any non-linear function arbitrarily, so that the neural network can be applied to most non-linear models.

However, it is also very important when we choose the activation function, because different activation functions have different effects on the convergence speed of the model and the training time. There are many activation functions, such as ELU (Exponential Linear Units) and ReLU (Rectified Linear Units), The ReLU function is a piecewise linear function that turns all negative values into 0, while the positive values remain unchanged. This method is called unilateral inhibition. With this unilateral inhibition, the nerves in the neural network can be Meta has sparse activation. But it also has its shortcomings. As the training deepens, neurons may die and the weights cannot be updated. Because the ReLU function can only output 0 and positive numbers, if a negative number is input, it will not be activated at all. The ReLU function is not a 0-centered function. If this happens, the gradient flowing through the neuron will always be 0 from this point on. The ELU function incorporates some properties of the ReLU function. The left side of the function has soft saturation, and the right side has no saturation. The average value of ELU output is basically close to 0, which makes it faster to converge. It reduces the gap between the normal gradient and the unit natural gradient, thereby speeding up the learning speed, and it can also be under negative constraints more robust [15]. But what we use here is the GELU activation function, which is what we often call the Gaussian error linear unit. The GELU activation function adds the idea of random regularization to the activation, which is equivalent to a probabilistic description of the neuron input. The non-linear change of the GLUE activation function is a random regular transformation method that meets expectations. Therefore, GlUE also has a high-performance activation function. The output images of the three activation functions are shown in **Figure 2**.

The GELU function we use here as the activation function of the output layer; the mathematical formula is as Formula (1):

$$\text{GELU}(x) = xP(X \le x) = x\Phi(x) \tag{1}$$

where $\Phi(x)$ refers to the cumulative distribution of the Gaussian normal distribution of *x*, as in Formula (2):

$$\Phi(x) = P(X \le x) = \int_{-\infty}^{x} \frac{e^{-\frac{(X-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma} \mathrm{d}X \tag{2}$$

The reason for choosing the GELU activation function formula is that according to the central limit theorem, the overall distribution of many independent random variables approximately obeys the normal distribution. Therefore, there are many situations in reality that can be modeled by an approximate normal distribution method, so it is more reasonable to use the normal distribution function as the activation function. Furthermore, among all possible distributions with the same variance, the normal distribution has the largest uncertainty, that is, the largest entropy.

In the fully connected layer, in addition to the activation function behind each layer, a Dropout layer is also added to randomly crop a certain proportion of neurons to prevent overfitting.

## 3.2. Adam Adaptive Moment Estimation Optimization

For the Adam algorithm, we must first understand the adaptive gradient algorithm (AdaGrad) and the root mean square propagation algorithm (RMSProp),

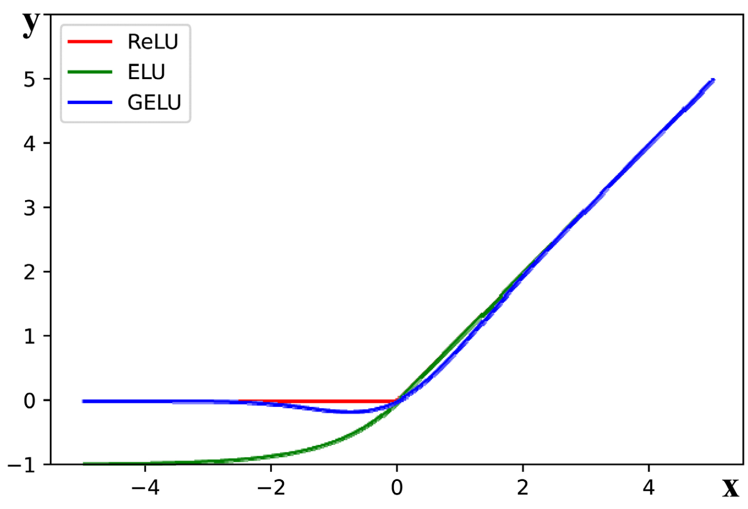

**Figure 2.** Three activation functions.

The basic idea of AdaGrad is to adaptively adjust its learning rate for each parameter. The adaptive method is to multiply each parameter by a different coefficient and this coefficient is determined by the sum of squares of the gradient size accumulated before. In other words, for those that have been updated a lot before, it can be relatively slow, and for those that have not been updated much, a larger learning rate can be given. The RMSProp is actually an improvement of AdaGrad, that is, it turns AdaGrad's sum of historical gradients into an average of historical gradients. Of course, this is not the mean in the strict sense. Then using this mean to replace the accumulated gradient of AdaGrad and weight the current gradient, and use it to update.

Assuming the loss function is as Formula (3):

$$Loss = x^2 + 4y^2 \tag{3}$$

That is, our goal is to learn the values of $x$ and $y$ to make the $Loss$ as small as possible. The drawing result of the loss function is shown in Figure 3.

Note that this is not a U-shaped slot. It has a minimum point. The $x$ and $y$ values corresponding to this point are the learning goals. Obviously when $x = 0$, $y = 0$, $Loss$ achieves the minimum value. But here we use neural network back propagation to find the derivation, and optimize the parameters step by step to make the $Loss$ smaller. Through this process, the function of RMSProp algorithm can be seen.

Adam's adaptive moment estimation algorithm has done gradient moving average and deviation correction based on RMSProp. In RMSProp, the square of the gradient is smoothed by a smoothing constant, but the gradient itself is not smoothed. In Adam, the gradient is smoothed, and the square gradient is also smoothed. The smoothed sliding averages are denoted by $\overrightarrow{m_t}$ and $\overrightarrow{v_t}$ respectively, and there are two $\beta$ in Adam. Assuming that at time *t*, the first derivative of the objective function with respect to the parameters is $g_t$, then the specific formula for calculating the gradient is as shown in Formula (4):

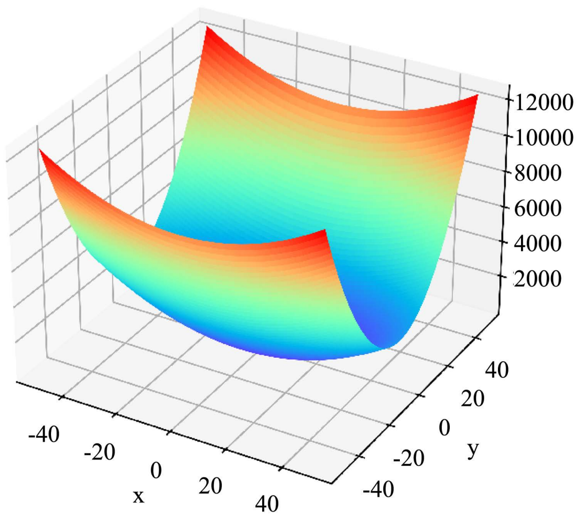

Figure 3. Loss function image.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)(g_t)^2 \end{aligned} \tag{4}$$

Next, calculate their respective sliding averages, the specific formula is as Formula (5):

$$\begin{aligned} \overrightarrow{m_t} &= \frac{m_t}{1-\beta_1^t} \\ \overrightarrow{v_t} &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{5}$$

The final gradient update method is as Formula (6):

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\overrightarrow{m_t}}{\sqrt{\overrightarrow{v_t}} + \varepsilon} \tag{6}$$

Among them, $\eta$ is the learning rate, $\beta_1$ is the exponential decay rate estimated for the first time, and $\beta_2$ is the exponential decay rate estimated for the second time, $\varepsilon = 10^{-8}$, The $\varepsilon$ in the denominator is to prevent Ho from being divided by 0 in implementation. In fact, for the learning rate, it is generally recommended to choose $\eta = 0.001$, $\theta_t$ is the last gradient, $\theta_{t+1}$ is the updated gradient. Note that *t* in $\beta_1^t$ and $\beta_2^t$ participates in exponential calculations. In fact, the current gradient update uses the exponential decay mean $\overrightarrow{v_t}$ of the square gradient $v_t$ at the previous moment and the exponential decay mean $\overrightarrow{m_t}$ of the gradient $m_t$ at the previous moment.

## 3.3. Weighted Cross-Entropy Loss Functıon Evaluation Algorithm

First of all, Cross-Entropy is an important concept in information theory, mainly used to measure the difference between two probability distributions. For the understanding of Cross-Entropy, we must firstly know what the amount of information is. For example, "there is sea in the sea", the amount of information in this sentence is 0, why? Because this is a nonsense, there must be sea water in the sea. Here is another one, such as "The new crown pneumonia epidemic will be completely over next year", Intuitively, this sentence has a lot of information, because the new crown pneumonia epidemic will end next year, there are great uncertainties, and this sentence eliminates the uncertainty of the new crown pneumonia epidemic ending next year. Therefore, by definition, this sentence is very informative. Of course, I'm just making an analogy. In summary, the probability of information occurrence is inversely proportional to the amount of information. The greater the probability, the smaller the amount of information. The smaller the probability, the greater the amount of information.

Suppose the probability of a certain event occurrence is $P(x)$, and its information content is expressed as shown in Formula (7):

$$I(x) = -\log_e(P(x)) \tag{7}$$

Among them, $I(x)$ represents the amount of information, and log represents the natural logarithm with e as the base.

The information entropy is also called entropy if you expect the amount of information. Expectation is the probability of possible outcomes in each experiment multiplied by the total number of outcomes. Therefore, the expression of information entropy is shown in Formula (8):

$$H(X) = -\sum_{i=1}^{n} P(x_i)\log_e\left(P(x_i)\right) \tag{8}$$

Here *X* is a discrete random variable, and *n* represents all *n* possibilities.

For the same random variable *X*, if there are two separate probability distributions, $P(x)$ and $Q(x)$, the difference between the two probability distributions can be measured by KL divergence. Such as Formula (9):

$$D_{KL}(p \parallel q) = \sum_{i=1}^{n} p(x_i)\log_e\left(\frac{p(x_i)}{q(x_i)}\right) \tag{9}$$

We further derive the KL divergence and simplify it as Formula (10):

$$\begin{aligned} D_{KL}(p \parallel q) &= \sum_{i=1}^{n} p(x_i)\log_e\left(\frac{p(x_i)}{q(x_i)}\right) \\ &= \sum_{i=1}^{n} p(x_i)\log_e\left(p(x_i)\right) - \sum_{i=1}^{n} p(x_i)\log_e\left(q(x_i)\right) \\ &= -H\left(p(x)\right) - \sum_{i=1}^{n} p(x_i)\log_e\left(q(x_i)\right) \end{aligned} \tag{10}$$

The former $H\left(p(x)\right)$ represents information entropy, and the latter is cross entropy as in Formula (11):

$$H(p,q) = -\sum_{i=1}^{n} p(x_i)\log_e\left(q(x_i)\right) \tag{11}$$

We use $p(x_i)$ to represent the true distribution of the sample, and $q(x_i)$ to represent the distribution predicted by the model.

In order to solve the problem of class imbalance in the data set, we attribute it to the imbalance in learning difficulty, which leads to different convergence speeds, so we thought of weighting in the loss function to balance the imbalance of samples in this way. So the Formula (12) is obtained:

$$\log H(p,q) = -\sum_{i=1}^{n} \omega_i p(x_i)\log_e\left(q(x_i)\right) \tag{12}$$

where $\omega_i$ represents the weight of the loss function when the actual label of the current data is.

## 3.4. Network Structure

Input the processed data into the deep neural network, use the fully connected neural network to extract the features of the data, and then use the Relu activation function and Gelu activation function to nonlinearize the output of the current layer in the same layer. Its structure is shown in the following Table 1.

Table 1. Network structure.

| Layer | Output shape | Activation | Connected to |
|---|---|---|---|
| Input1 | (None, 41) | None | |
| Dense1 | (None, 128) | Relu | Input1 |
| Dense2 | (None, 128) | Gelu | Input1 |
| Concatenate1 | (None, 256) | None | Dense1<br>Dense2 |
| Dropout1 | (None, 256) | None | Concatenate1 |
| Dense3 | (None, 768) | Relu | Dropout1 |
| Dense4 | (None, 768) | Gelu | Dropout1 |
| Concatenate2 | (None, 1536) | None | Dense3<br>Dense4 |
| Dropout2 | (None, 1536) | None | Concatenate2 |
| Dense5 | (None, 150) | Relu | Dropout2 |
| Dense6 | (None, 150) | Gelu | Dropout2 |
| Concatenate3 | (None, 300) | None | Dense5<br>Dense6 |
| … | … | … | … |
| Concatenate7 | (None, 256) | None | Dense11<br>Dense12 |
| Dense13 | (None, 1) | Sigmoid | Concatenate7 |

## 4. Intrusion Detection System Design

First of all, our detection model has a data acquisition and processing module, an intrusion detection module, a detection classification module, and a visual analysis module.

Data collection and processing: Obtaining the network data set, performing preprocessing operations such as feature extraction, numerical conversion, and data normalization on the network data set, then checking the numerical data distribution and dividing it into a test set and a training set, which are used for model testing and training respectively.

Intrusion detection module: determining the input and output nodes of the deep neural network according to the dimensions of the preprocessed data, then determining the entire network structure and training parameters according to the hidden layer and other parameters, using the training set to train the model, and saving the model for testing after completing the training.

Detection and classification module: testing the test set and classifying the test results.

Visual analysis module: Visually display the distribution of numerical data and the classification results, and then make an analysis. The model structure diagram is shown in Figure 4.

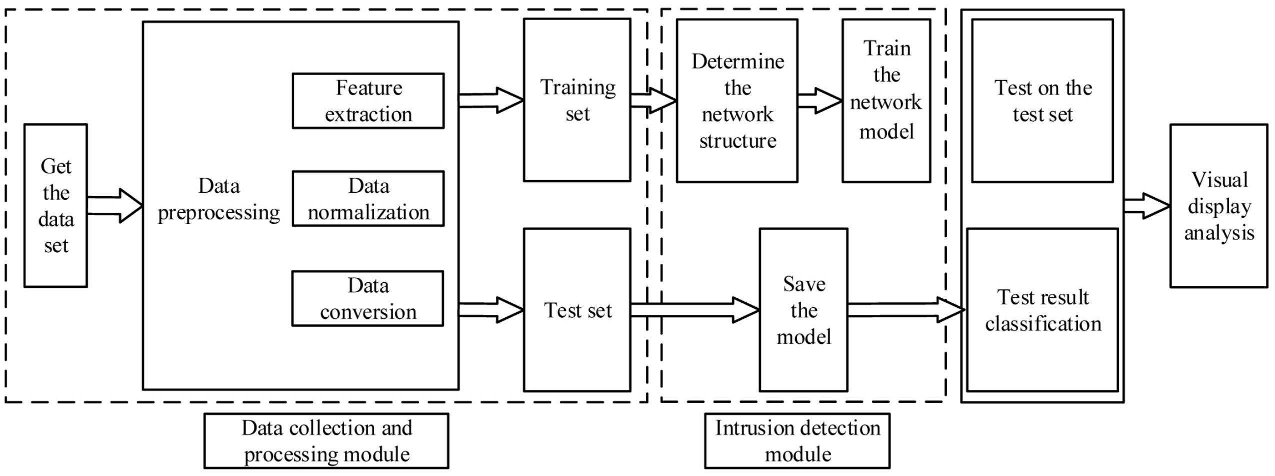

Figure 4. Intrusion detection system diagram.

## 4.1. Data Set Selection

Here we have selected the KDDCup99 data set, which is more common in intrusion detection, for the convenience of comparison experiments. The data set has 42 dimensions, of which 41 dimensions are attributed features, and 1 dimension is flag feature. The release of the KDD Cup99 data set is very useful for many IDS evaluations, and it is also a widely used data set. The data set is composed of 5 million network connection records containing 41 characteristics. The simulated attacks can be divided into 4 categories:

**Denial of service attack (DOS):** The intruder exhausts the resources of the attacked object by attacking the defects realized by the network protocol or directly using brute force. The purpose is to make the target computer or network unable to provide normal service or resource access, so that the target system service system stops responding or even crashes, thereby causing service interruption.

**Port monitoring or scanning attack (Probe):** The network intruder collects information about the types of computers on the network, and then gains root access through the firewall of the target host.

**Remote to Local Attack (R2L):** The network intruder sends data packets to the target, but does not have a user account on the host itself, trying to use the vulnerability to gain local access, pretending to be an existing user of the target host.

**User to Root Attack (U2R):** A commonly used method of network intrusion, the intruder tries to take advantage of the user's pre-existing access rights and exploits loopholes to gain root control.

Due to the huge amount of data and the limitation of memory allocation, we use 10% of the actual amount of data here. Then here we use Numpy in python to perform statistical data to get the following data set data distribution table as shown in Table 2.

The KDD data set has a total of 41 attribute features and 1 logo feature. The specific information is shown in Table 3.

Table 2. Data distribution.

| Normal | DOS | Probe | R2L | U2R | Total |
|---|---|---|---|---|---|
| 97,278 | 391,458 | 4107 | 1126 | 52 | 494,021 |

Table 3. Attribute information of KDD Dataset.

| Column label number | Attribute characteristics |
|---|---|
| 1 - 9 | Basic characteristics of network connection |
| 10 - 22 | Content characteristics of network connections |
| 23 - 31 | Time-based flow characteristics |
| 32 - 41 | Host-based traffic characteristics |
| 42 | Logo feature |

## 4.2. Data Preprocessing

**1) Numerical Processing**

For symbolic features, we use one-hot code, which is, there are as many bits as there are states, and only one bit is 1, and the others are all 0. For example, the Normal code is 10,000. For character data, it is converted to numeric data. When transforming data, the method of function mapping is adopted, and each type of character form corresponds to a uniquely determined binary code, which is, in the formula: is the original character string in the network stream data feature, is the data in binary encoding format; is the mapping relationship.

**2) Standardization**

First of all, ordinary standardization is to calculate the average value $\overline{x_k}$ and the average absolute error $S_k$ of each attribute. The calculation formula is as Formula (13):

$$\begin{aligned} \overline{x_k} &= \frac{1}{n}\sum_{i=1}^{n} x_{ik} \\ S_k &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_{ik}-\overline{x_k}\right)^2} \end{aligned} \tag{13}$$

where $x_{ik}$ represents the *k*-th attribute of the *i*-th record, $S_k$ represents the average absolute error of the *k*-th attribute, $\overline{x_k}$ represents the mean value of the *k*-th attribute. Then standardize the measurement for each data record, such as Formula (14):

$$Z_{ik} = \frac{x_{ik}-\overline{x_k}}{S_k} \tag{14}$$

Among them, $Z_{ik}$ represents the *k*-th attribute value of the *i*-th record after standardization. However, adding Z-Score here is equivalent to doing another calculation after normal standardization, which is actually a process of dividing the difference between the score and the average by the standard deviation. Converting the raw scores in the normally distributed data to Z-Scores, we can know the area between the average and the Z-score by consulting the table of the area under the normal curve of the Z-score, and then know the percentage rank of the original score in the data set. Z-Score is a way to see the relative position of a certain score in the distribution. The specific formula is as Formula (15):

$$z = \frac{x - \mu}{\sigma} \tag{15}$$

Among them, $\mu$ is the mean value of all data, $\sigma$ is the standard deviation, $x$ is the original data, and the $z$ value represents the distance between the original score and the population average, and is calculated in the unit of standard deviation.

**3) Normalization**

In fact, each value after standardization is normalized to the interval [0, 1]. Its formula is as Formula (16):

$$x^{\otimes} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{16}$$

where $x_{\min}$ and $x_{\max}$ are the minimum and maximum values of each data item, $x$ is the value of the original data, and $x^{\otimes}$ is the normalized data.

**4) Divide the Data Set**

After the data is preprocessed, 20% is randomly selected as the test set, and the remaining 80% is used as the training set. The data after the split is shown in Table 4.

## 5. Experiment and Analysis

### 5.1. Lab Environment

In order to build the model and train the parameters smoothly and effectively in the intrusion detection algorithm experiment, we use the Keras deep learning framework of TensorFlow. The specific hardware environment and software environment of the experiment are shown in Table 5.

### 5.2. Data Analysis

**1) Attack Type Exploration**

Firstly, we will subdivide the statistics of the 4 commonly used attack types in the data set. As shown in Table 6.

Then we add these attack types and "Normal" types to the dictionary to match the predicted attack column "target". Map the class name according to the

Table 4. Data set division.

| Data set | Total amount of data | Independent feature type | Dependent feature type |
|---|---|---|---|
| KDDCup99 | 494,021 | 122 | 5 |
| Training set | 395,216 | 122 | 5 |
| Test set | 98,805 | 122 | 5 |

Table 5. Lab Environment.

| Name | Configuration |
|---|---|
| CPU | AMD Ryzen 5 3600 6-Core |
| GPU | NVIDIA GeForce RTX 2060 |
| RAM | 16GB |
| Operating System | Windows10 Professional |
| Programming language | Python3.9 |
| Visual analysis tools | Matplotlib |

Table 6. Attack breakdown statistics.

| Attack type | DOS | Probe | R2L | U2R |
|---|---|---|---|---|
| 1 | back | ipsweep | ftp_write | buffer_overflow |
| 2 | land | portsweep | guess_passwd | loadmodule |
| 3 | neptune | satan | imap | perl |
| 4 | pod | | multihop | rootkit |
| 5 | smurf | | phf | |
| 6 | teardrop | | spy | |
| 7 | | | warezclient | |

column where the predicted attack is located, use the value counts() function to check the unique value in the target column and visually display the number of repetitions of each tag in the predicted attack. As shown in Figure 5.

Here we find that there is an extra "." at the end of each attack name. Therefore, here we use this format to match the actual attack type. Map the actual attack type to another column named "target_type", and visually display the actual attack type statistics as shown in Figure 6.

**2) Classification Feature Exploration**

Here we use the info() function in python to check whether there are missing values in each column of the data set. We found no data loss. Then we get the names of all numeric columns as "target_type", "service", "flag", "target", "protocol_type", noting that the "target" column here is our prediction, "Target_type" is packet data. Then we are determining whether there is any other binary data. We found that there is also "land", "logged_in", "root_shell", "num_outbound_cmds", "is_host_login", "is_guest_login", The meanings they represent are as follows.

**3) Digital Feature Exploration**

Identify the remaining digital features by subtracting the classification column.

Here we use the standard deviation to measure their degree of deviation. Standard deviation is a measure of the degree of dispersion of data distribution,

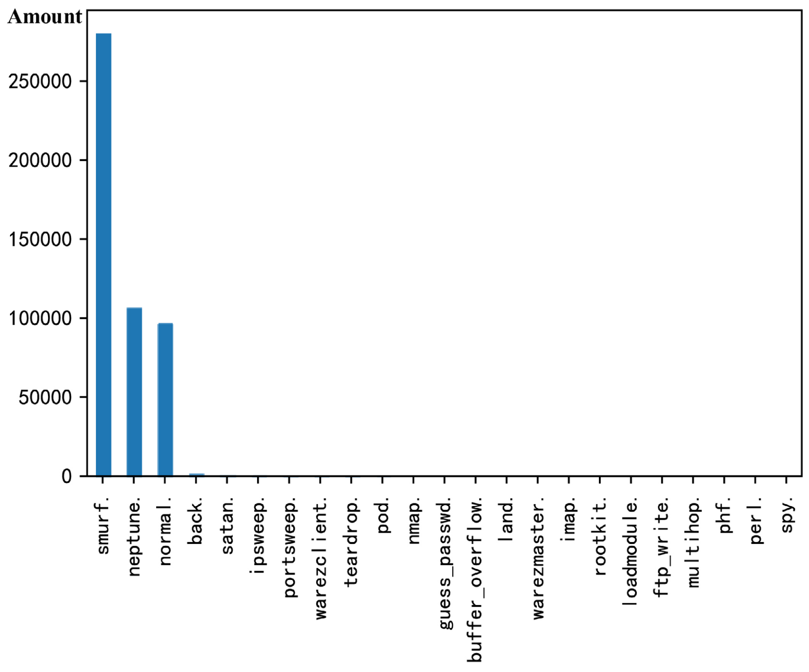

**Figure 5.** Repetition degree of each label in predicted attack.

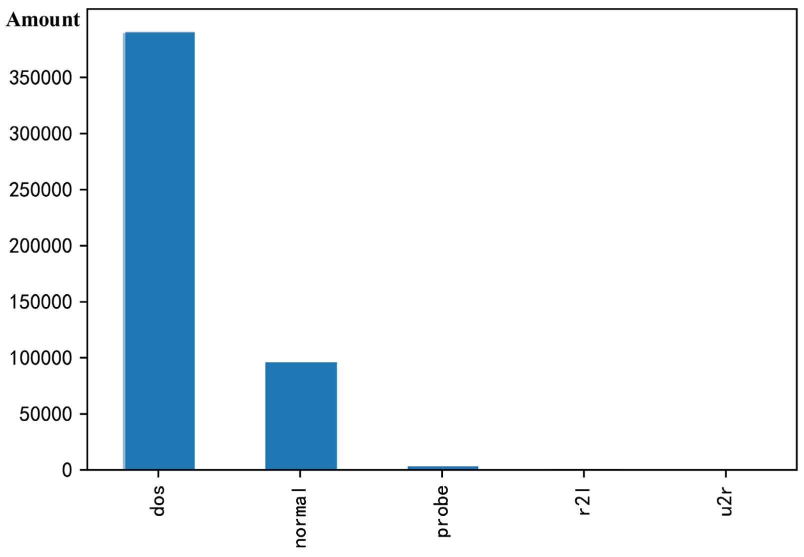

**Figure 6.** Statistics of actual attack types.

being used to measure the degree of deviation of the data value from the arithmetic mean. The smaller the standard deviation, the less these values deviate from the average, and vice versa. The size of the standard deviation can be measured by the magnification relationship between the standard deviation and the average value. Then we visualize their standard deviations as shown in Figure 7.

Because there are different normal distributions in the figure, we are applying Z-Score to further standardize the digital features, after further standardization, as shown in Figure 8.

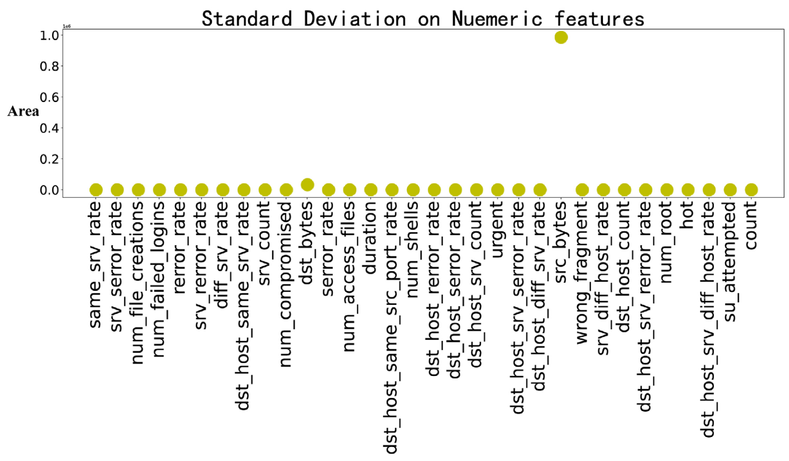

**Figure 7.** Standard deviation of number features.

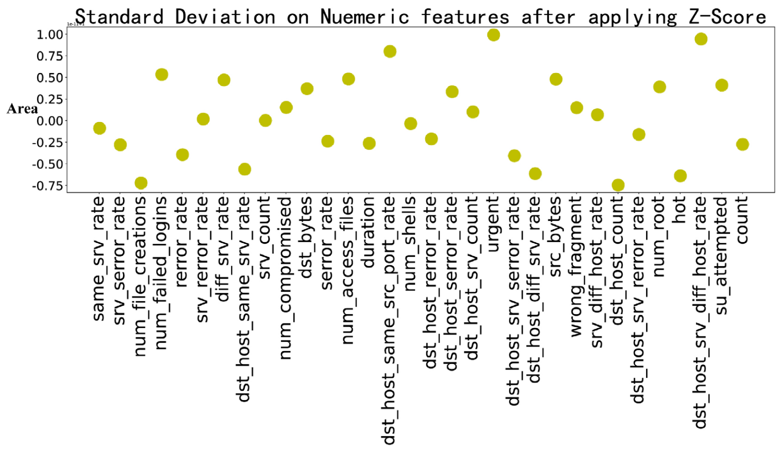

**Figure 8.** Standard deviation on nuemeric features after applying z-score.

## 5.3. Experimental Data Comparison

This article uses Accuracy, Precision, Recall, F1-Score to evaluate the model. The formula of the four parameters is as the Formulas (17) - (20):

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{17}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{18}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{19}$$

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{20}$$

Among them, (True Positive) represents the number of samples that represent the attack as an attack type, (True Negative) represents the number of samples that judge the attack type as a normal type, (False Positive) represents the number of samples that judge a normal sample as an attack type, (False Negative) represents the number of samples that define the attack as a normal type.

The following table shows all the parameter settings of the loss function algorithm experiment after the Epoch of the experiment is determined, as shown in Table 7.

Below we compare and analyze the experimental data between the ordinary cross-entropy loss function model and the weighted cross-entropy loss function model. The data is shown in Table 8 and Figure 9.

From the above data comparison table and comparison chart analysis, the weighted cross entropy loss function is significantly better than the ordinary cross entropy loss function in terms of accuracy and various numerical values.

Let's look at the weighted Cross-Entropy loss function training data experiment. The experimental parameters are given above. Let us directly look at the experimental data table, as shown in Table 9.

Here due to the use of the Early stopping method, when we train deep learning neural networks, we usually hope to get the best generalization performance,

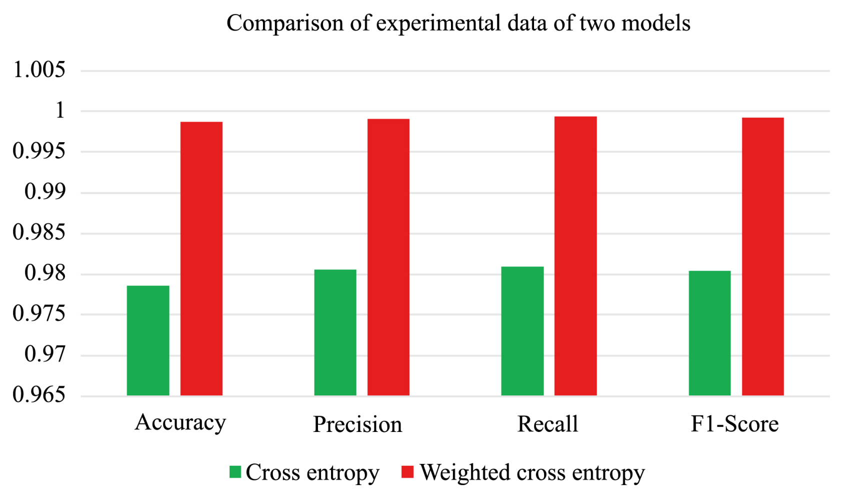

**Figure 9.** Comparison of experimental data of two models.

**Table 7.** Model experiment parameter setting.

| Parameter Name | Parameter |
|---|---|
| Activation function | Relu, Gelu |
| Gradient descent optimizer | Adam |
| Initial learning rate | 0.001 |
| Batch size | 200 |
| Epoch | 50 |

**Table 8.** Comparison of experimental data of two models.

| Model | Accuracy | Precision | Recall | F1Score |
|---|---|---|---|---|
| Cross entropy | 0.9786 | 0.9806 | 0.9809 | 0.9804 |
| Weighted Cross entropy | 0.9991 | 0.9995 | 0.9998 | 0.9996 |

Table 9. Weighted cross-entropy loss function algorithm experimental data table.

| Epoch | Time cost | Loss of train | Accuracy of train | Loss of test | Accuracy of test |
|---|---|---|---|---|---|
| 1/50 | 9s | 0.1344 | 0.9906 | 0.0197 | 0.9963 |
| 2/50 | 8s | 0.0157 | 0.9967 | 0.0144 | 0.9970 |
| 3/50 | 8s | 0.0120 | 0.9970 | 0.0140 | 0.9969 |
| 4/50 | 8s | 0.0105 | 0.9974 | 0.0119 | 0.9970 |
| 5/50 | 8s | 0.0096 | 0.9978 | 0.0110 | 0.9983 |
| 6/50 | 8s | 0.0091 | 0.9983 | 0.0112 | 0.9972 |
| 7/50 | 8s | 0.0087 | 0.9986 | 0.0122 | 0.9983 |
| 8/50 | 8s | 0.0083 | 0.9987 | 0.0129 | 0.9982 |
| 9/50 | 8s | 0.0084 | 0.9989 | 0.0115 | 0.9989 |
| 10/50 | 8s | 0.0084 | 0.9988 | 0.0113 | 0.9986 |
| 11/50 | 8s | 0.0078 | 0.9989 | 0.0127 | 0.9981 |
| 12/50 | 8s | 0.0077 | 0.9990 | 0.0123 | 0.9989 |
| 13/50 | 8s | 0.0079 | 0.9990 | 0.0123 | 0.9989 |
| 14/50 | 8s | 0.0079 | 0.9991 | 0.0111 | 0.9990 |
| 15/50 | 8s | 0.0078 | 0.9990 | 0.0111 | 0.9988 |
| 16/50 | 8s | 0.0074 | 0.9990 | 0.0110 | 0.9989 |
| 17/50 | 8s | 0.0076 | 0.9990 | 0.0100 | 0.9989 |
| 18/50 | 8s | 0.0073 | 0.9990 | 0.0108 | 0.9989 |
| 19/50 | 8s | 0.0073 | 0.9990 | 0.0110 | 0.9990 |
| 20/50 | 8s | 0.0071 | 0.9991 | 0.0105 | 0.9991 |
| 21/50 | 8s | 0.0071 | 0.9991 | 0.0101 | 0.9990 |
| 22/50 | 8s | 0.0072 | 0.9991 | 0.0097 | 0.9991 |

that is, we can fit the data well. But all standard deep learning neural network structures such as fully connected multilayer perceptrons are easy to overfit. That is, when the network performs better and better on the training set, and the error rate is getting lower and lower. In fact, at a certain moment, its performance on the test set has begun to deteriorate. In order to prevent overfitting, we use this method.

The results of the visual analysis are shown in Figure 10 and Figure 11.

It can be seen from the above that after the model is trained, the accuracy curve is in a relatively balanced state, which shows that the fluctuation range of the model is not large and relatively stable during the training process.

Then we experimentally compare the data of this model with other models, as shown in Table 10 and Figure 12.

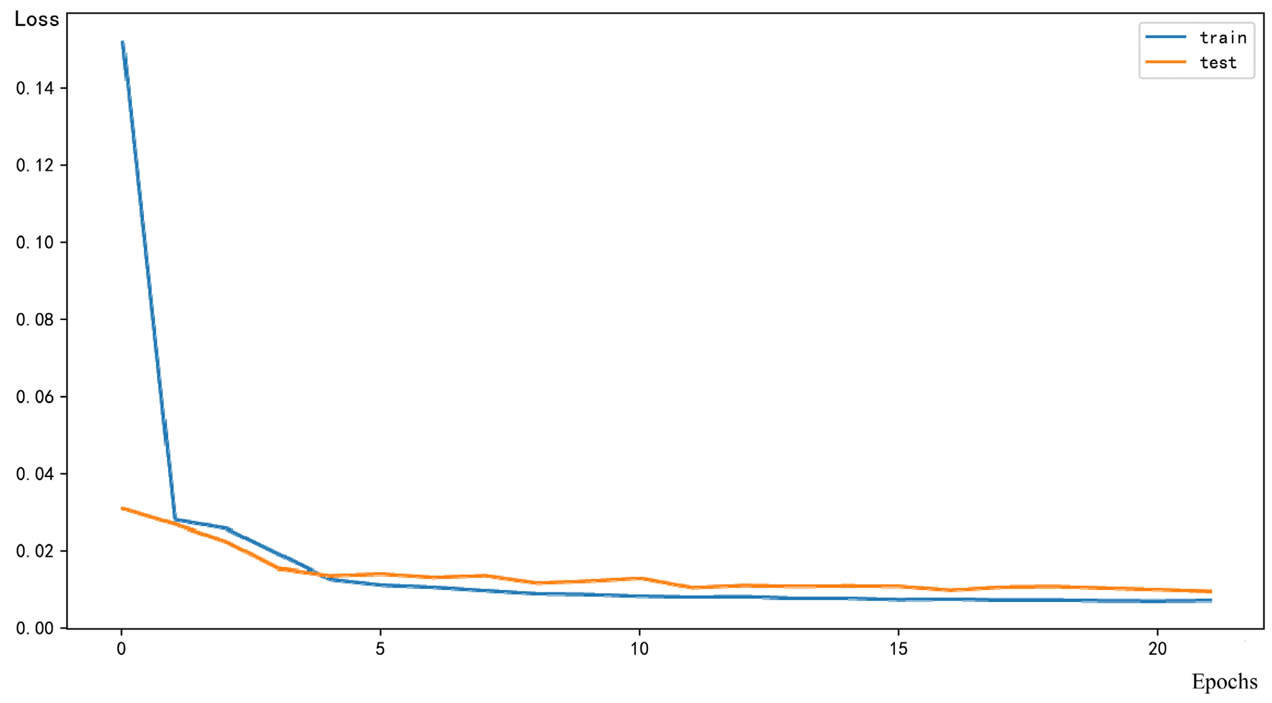

**Figure 10.** The loss rate under the weighted cross-entropy loss function model.

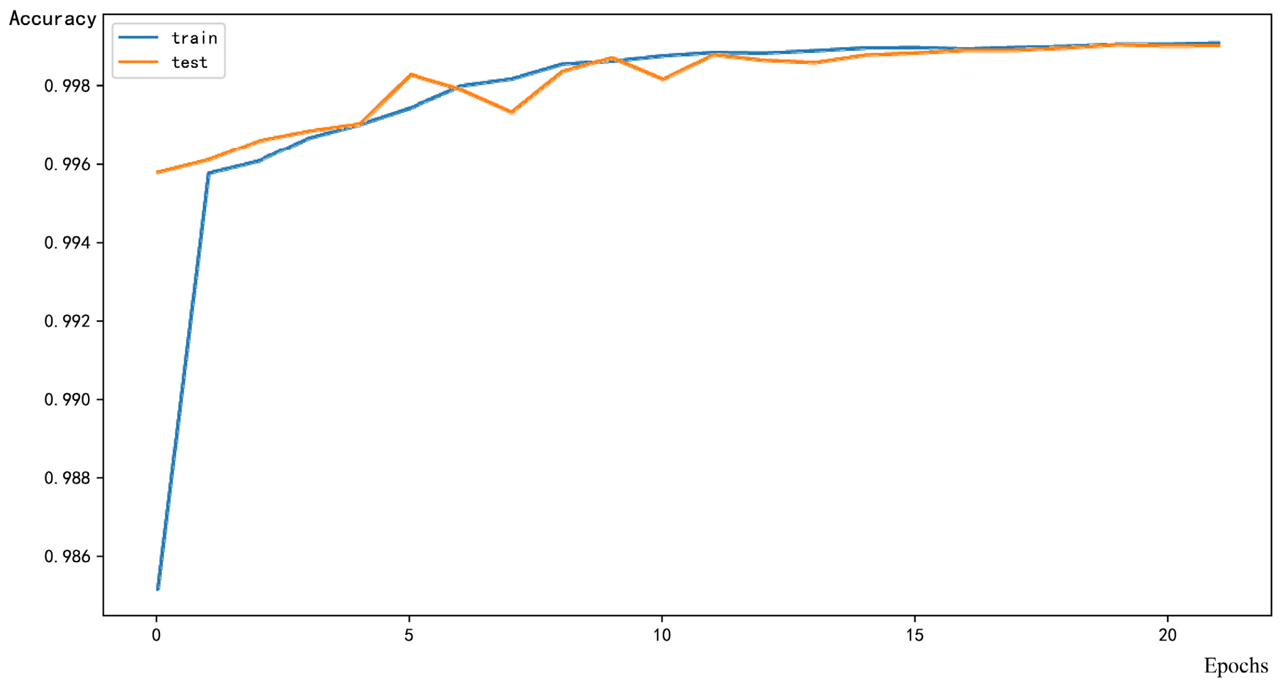

**Figure 11.** The accuracy of the weighted cross-entropy loss function model.

From the above data, it can be seen that the model has a certain improvement in data than other models, but this may be due to the overfitting phenomenon caused by the excessively strong model training due to the problem of gradient optimization, but from the comparison of this data, The accuracy rate has indeed improved.

**Table 10.** Comparison of experimental data of various models.

| Model | Accuracy | Precision | Recall | F1Score |
|---|---|---|---|---|
| CNN | 0.9904 | 0.9954 | 0.9909 | 0.9930 |
| ANN | 0.9786 | 0.9834 | 0.9839 | 0.9833 |
| KNN | 0.9826 | 0.9874 | 0.9879 | 0.9876 |
| DNN | 0.9834 | 0.9886 | 0.9887 | 0.9884 |
| DNN + Gelu | 0.9987 | 0.9991 | 0.9994 | 0.9992 |

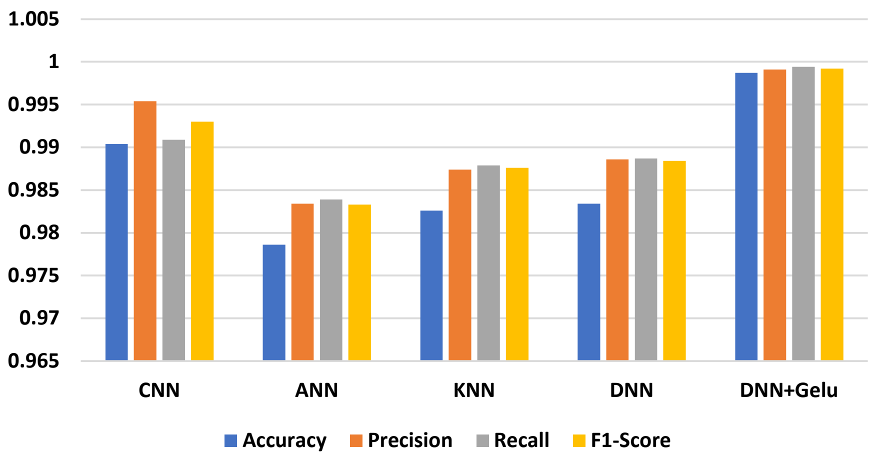

**Figure 12.** Comparison of experimental data of various models.

## 6. Conclusion

Using the DNN + Gelu model architecture, the cross-entropy function is improved to a weighted cross-entropy loss function, a new intrusion detection system is constructed and applying a weighted loss function to improve the accuracy of model. In order to prove the role of the weighted loss weight function, this paper compares and analyzes with other models based on a commonly used intrusion detection data set KDDCup99, which will be more convincing. After data analysis, it is proved that the weighted loss weighted function can improve the accuracy of model recognition. However, the batch_size and epoch trained here are relatively fixed. If you change the training accuracy of these variables, it remains to be tested, and the choice of optimizer may also affect the training accuracy of the model. These are the problems that this article will solve later.

## Fund

This work was Supported by Sichuan Science and Technology Program (2020YFG0151), The Opening Project of Key Laboratory of Higher Education of Sichuan Province for Enterprise Informationalization and Internet of Things (2021WZY01) and Postgraduate Innovation Fund Project of Sichuan University of Science and Engineering (y2021091).

## Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

## References

[1] Eesa, A.S., Orman, Z. and Brifcani, A.M.A. (2015) A New Feature Selection Model Based on ID3 and Bees Algorithm for Intrusion Detection System. *Turkish Journal of Electrical Engineering & Computer Sciences*, **23**, 615-622. https://doi.org/10.3906/elk-1302-53

[2] Abdulqader, D.M., Abdulazeez, A.M. and Zeebaree, D.Q. (2020) Machine Learning Supervised Algorithms of Gene Selection: A Review. *Machine Learning*, **62**, 233-244.

[3] Xiong, J., Qin, R., He, M., Liu, J. and Tang, F. (2021) Application of Improved Random Forest Algorithm in Android Malware Detection. *Computer Engineering and Applications*, **57**, 130-136. (in Chinese)

[4] LeCun, Y., Yoshua, B. and Geoffrey, H. (2015) Deep Learning. *Nature*, **521**, 436-444. https://doi.org/10.1038/nature14539

[5] Giuseppina, A., Appice, A. and Malerba, D. (2021) Nearest Cluster-Based Intrusion Detection through Convolutional Neural Networks. *Knowledge-Based Systems*, **216**, Article ID: 106798. https://doi.org/10.1016/j.knosys.2021.106798

[6] Michał, C. and Pawlicki, M. (2021) Intrusion Detection Approach Based on Optimised Artificial Neural Network. *Neurocomputing*, **452**, 705-715. https://doi.org/10.1016/j.neucom.2020.07.138

[7] Soosan Naderi, M. and Kahani, M. (2021) A Novel Scalable Intrusion Detection System Based on Deep Learning. *International Journal of Information Security*, **20**, 387-403. https://doi.org/10.1007/s10207-020-00508-5

[8] Mendonça, R.V., Teodoro, A.A.M., Rosa, R.L., Saadi, M., Carrillo Melgarejo, D., Nardelli, P.H.J., *et al.* (2021) Intrusion Detection System Based on Fast Hierarchical Deep Convolutional Neural Network. *IEEE Access*, **9**, 61024-61034. https://doi.org/10.1109/ACCESS.2021.3074664

[9] Giuseppina, A., Appice, A. and Malerba, D. (2021) Autoencoder-Based Deep Metric Learning for Network Intrusion Detection. *Information Sciences*, **569**, 706-727. https://doi.org/10.1016/j.ins.2021.05.016

[10] Khan, M.A. (2021) HCRNNIDS: Hybrid Convolutional Recurrent Neural Network-Based Network Intrusion Detection System. *Processes*, **9**, Article No. 834. https://doi.org/10.3390/pr9050834

[11] Sajith, P.J. and Nagarajan, G. (2021) Optimized Intrusion Detection System Using Computational Intelligent Algorithm. In: Mallick P.K., Bhoi A.K., Chae G.S. and Kalita K., Eds., *Advances in Electronics*, *Communication and Computing*, Springer, Singapore, 633-639. https://doi.org/10.1007/978-981-15-8752-8_64

[12] Srinidhi, C.L., Ozan, C. and Martel, A.L. (2021) Deep Neural Network Models for Computational Histopathology: A Survey. *Medical Image Analysis*, **67**, Article ID: 101813. https://doi.org/10.1016/j.media.2020.101813

[13] Tian, P., Chen, Z., Yu, W. and Liao, W. (2021) Towards Asynchronous Federated Learning Based Threat Detection: A DC-Adam Approach. *Computers & Security*, **108**, Article ID: 102344. https://doi.org/10.1016/j.cose.2021.102344

[14] Bihonegn, T., Kaushik, S., Bansal, A., Vojtíšek, L. and Slovák, J. (2021) Geodesic Fiber Tracking in White Matter Using Activation Function. *Computer Methods and*

*Programs in Biomedicine*, **208**, Article No. 106283.
https://doi.org/10.1016/j.cmpb.2021.106283

[15] Cococcioni, M., Rossi, F., Ruffaldi, E. and Saponara, S. (2020) A Novel Posit-Based Fast Approximation of elu Activation Function for Deep Neural Networks. 2020 *IEEE International Conference on Smart Computing* (*SMARTCOMP*). Bologna, 14-17 September 2020, 244-246.
https://doi.org/10.1109/SMARTCOMP50058.2020.00053