Sensors and Materials, Vol. 37, No. 5 (2025) 2091–2104
MYU Tokyo

S & M 4044

Distributed Denial of Service Attack Detection Based on Cuckoo Search Bidirectional Learning Method

Hongxiang Ke,[1] Huoyou Li,[2*] and Cheng-Fu Yang[3,4**]

[1]Zhangzhou College of Science and Technology, Zhangzhou 363200, China
[2]School of Mathematics and Information Engineering, Longyan University, Longyan 364012, China
[3]Department of Chemical and Materials Engineering, National University of Kaohsiung, Kaohsiung 811, Taiwan
[4]School of Energy and Power Engineering, Guangdong University of Petrochemical Technology, Maoming, Guangdong Province 525000, China

(Received January 27, 2025; accepted April 21, 2025)

Keywords: distributed denial of service, attack detection, bidirectional long short-term memory, cuckoo search bidirectional learning method, particle swarm optimization model

Distributed denial of service (DDoS) attacks pose a critical network security threat by exhausting server resources and bandwidth, rendering systems incapable of delivering essential services. While bidirectional long short-term memory (BLSTM) neural networks can detect these attacks, the bidirectional learning (BL) model, despite its suitability for handling large-scale and multi-attribute datasets, suffers from temporal interdependence limitations and suboptimal performance. In this paper, we introduced the cuckoo search (CS) bidirectional learning method (CSBLM), an innovative optimization model that enhanced the BL network performance through dynamic parameter tuning. At its core, CSBLM leveraged an optimized CS algorithm to fine-tune crucial BL neural network parameters, dynamically optimizing both the number of hidden units in the LSTM layer and the ideal time series length for processing. This sophisticated parameter optimization strategy represents a significant advancement in DDoS attack detection methodology. Experimental results demonstrated CSBLM's superior performance compared with conventional optimization approaches, including gray wolf and particle swarm optimization models. The implementation of CSBLM achieved outstanding results, significantly reducing the number of network operation iterations while enhancing detection accuracy to an impressive 99.09%. These outcomes firmly established CSBLM as a powerful solution for improving DDoS attack detection, offering both enhanced efficiency and exceptional accuracy in identifying and mitigating network security threats.

1. Introduction

The development of internet technologies has also led to a variety of distributed denial of service (DDoS) attack methods. Classic attack types include synchronize (SYN) flood, domain name system (DNS) query flood, internet control message protocol (ICMP) flood, user datagram protocol (UDP) flood, and network time protocol flood, among others.(1) During an attack,

*Corresponding author: e-mail: lhy@lyun.edu.cn
**Corresponding author: e-mail: cfyang@nuk.edu.tw
https://doi.org/10.18494/SAM5567

ISSN 0914-4935 © MYU K.K.
https://myukk.org/

attackers often use a combination of techniques to achieve different objectives. The aim of some attacks, such as ICMP flood and UDP flood, is to consume bandwidth by exploiting network transmission protocols. Others, like SYN flood, target the resources of the victim server by continuously initiating requests, leading to central processing unit (CPU) overload or memory exhaustion. Over one-fifth of businesses worldwide are currently affected by DDoS attacks, and this number continues to rise every year.[2] In February 2018, GitHub, an online code management service, was targeted by a Memcached DDoS attack. During the attack, the website's traffic surged to 1.3 Tbps.[3] However, owing to GitHub's strong focus on DDoS protection, they were able to prevent significant damage. In February 2020, Amazon web services, including simple storage service and other services, experienced a DDoS attack. This attack caused an 8-hour outage, making it one of the largest DDoS attacks on record, with a capacity of 2.3 Tbps.[4] The attack also crippled DNS web routers, impacting other services such as elastic load balancing, relational database service, and elastic compute cloud, which are commonly used for querying public DNS resolution systems.[5]

DDoS attacks primarily take two forms.[6] One type targets the network traffic, overwhelming the bandwidth with attack packets and preventing legitimate traffic from reaching the server. The other targets system resources, such as memory and CPU, by sending excessive requests that disable the server's ability to respond. DDoS attacks can be classified into direct and reflection attacks.[7] Direct attacks send a high volume of packets, such as transmission control protocol (TCP), ICMP, and UDP packets. A common method is TCP SYN flooding, which exploits the TCP three-way handshake.[8] In this attack, massive TCP SYN packets are sent to the server, which responds to each packet, but since the source IP is random, connections cannot be established; consequently, resources are drained. Reflection attacks use intermediate machines, like routers or printers, to send requests to the target server under the attacker's command.[9] If enough machines are involved, the reflection packets flood the target's network link. To address DDoS attacks, researchers have employed various approaches.

Li *et al.* developed a mathematical model using queuing theory in a container-based cloud environment to model low-rate distributed denial-of-service (LDoS) attacks.[10] Shon *et al.* used a genetic algorithm (GA) to select features and a support vector machine (SVM) for DDoS detection.[11] Yuan *et al.* proposed the use of deep learning models to learn high-order features of DDoS data, leveraging deep networks and historical network data to overcome the high error rates of shallow machine learning.[12] Jin *et al.* proposed a hybrid model, DCNN_DSAE, which combines the strengths of a deep convolutional neural network and deep stacked autoencoder (DSAE) algorithms to improve both accuracy and efficiency.[13] Bhardwaj *et al.* integrated a DSAE for feature learning with a deep neural network to classify network traffic into benign and DDoS attack traffic.[5] Peng *et al.* explored the effect of hyperparameters on neural network performance by using two structures: basic neural networks (BNNs) and long short-term memory recurrent neural networks (LSTM RNNs).[1] They applied grid search algorithms to optimize learning rates and number of iterations, minimizing the cost function of BNNs and transferring the optimized parameters to the LSTM network.

Sensors are typically used to collect network traffic, data packets, and various system operation parameters that serve as the foundation for attack detection. Sensors can be deployed

within the network to continuously monitor traffic and detect anomalous activities, such as signs of DDoS attacks. The optimized model proposed in this paper [such as the cuckoo search (CS) bidirectional learning method (CSBLM)] is aimed at improving the accuracy, scope, and ease of implementation of DDoS attack detection. This means that when implementing these models, the data provided by the sensors (e.g., network traffic, connection status, etc.) will be used for more precise analysis. Given the challenges mentioned above, there is a clear need for a more comprehensive, accurate, and easily implementable method for detecting DDoS attacks. The optimized model proposed in this paper effectively addresses these issues. We introduce an optimized approach, the CSBLM, which combines improved accuracy, a broader scope, and simpler implementation for DDoS detection. The main contributions of this paper are as follows. First, we present a dynamic optimization technique for the core parameters of the bidirectional learning (BL) neural network using an enhanced CS algorithm. Second, we develop the CSBLM detection algorithm. Finally, we conduct comparative experiments of CSBLM with traditional models, such as grey wolf optimization (GWO) and particle swarm optimization (PSO), to validate its effectiveness.

2. Related Principles and Technologies

2.1 Bidirectional LSTM network

The BL network is a type of recurrent neural network designed to address the vanishing gradient problem that occurs in deeper network structures.[14] This is achieved through gating units that set thresholds, enabling the BL recurrent unit to retain information for long periods. The BL recurrent unit consists of three gates: the forget gate, input gate, and output gate. The forget gate discards or forgets information from previous time steps, the input gate reads and selects information from the current time step, and the output gate determines which information is output from the current cell. The forward propagation of the BL is expressed as

$$\begin{aligned} \tau_f^t &= \sigma\left(W_f\left[a^{t-1}, x^t\right] + b_f\right), \quad \tau_u^t = \sigma\left(W_u\left[a^{t-1}, x^t\right] + b_u\right), \\ \tilde{c}^t &= \tanh\left(W_c\left[a^{t-1}, x^t\right] + b_c\right), \quad c^t = \tau_f^t \circ c^{t-1} + \tau_u^t \circ \tilde{c}^t, \\ \tau_o^t &= \sigma\left(W_o\left[a^{t-1}, x^t\right] + b_o\right), \quad a^t = \tau_o^t \circ \tanh\left(c^t\right), \end{aligned} \tag{1}$$

where τ_f, τ_u, and τ_o represent the forget gate, input gate, and output gate, respectively; $\tilde{c}^t$ represents the candidate value of the memory cell; and τ_u can be used to determine whether to use $\tilde{c}^t$ to update c^t. σ represents the sigmoid function, and $\tanh(c^t)$ is commonly used for linear activation functions. a^t represents the activation value at time t. After forward propagation, backpropagation is performed, and the gradient used to update each parameter is represented by Eq. (2), where $d\tau_f^t$, $d\tau_u^t$, and $d\tau_o^t$ represent the update gradients of the three gates.

$$\begin{aligned} d\tau_f^t &= \left(dc^t + da^t * \tau_o^t * \left[1 - \tanh^2\left(c^t\right)\right]\right) * c^{t-1} * \tau_f^t * \left(1 - \tau_f^t\right) \\ d\tau_u^t &= \left(dc^t + da^t * \tau_o^t * \left[1 - \tanh^2\left(c^t\right)\right]\right) * \tilde{c}^t * \tau_u^t * \left(1 - \tau_u^t\right) \\ d\tau_o^t &= da^t * \tanh\left(c^t\right) * \tau_o^t * \left(1 - \tau_o^t\right) \end{aligned} \tag{2}$$

dW_f, dW_u, dW_c, and dW_o represent the updated gradient of weight evidence, as shown below.

$$\begin{aligned} dW_f &= d\tau_f^t * \left(\left[a^{t-1}, x^t\right]^T\right) \quad dW_u = d\tau_u^t * \left(\left[a^{t-1}, x^t\right]^T\right) \\ dW_c &= d\tilde{c}^t * \left(\left[a^{t-1}, x^t\right]^T\right) \quad dW_o = d\tau_o^t * \left(\left[a^{t-1}, x^t\right]^T\right) \end{aligned} \tag{3}$$

dc^{t-1}, da^{t-1}, and dx^t respectively represent the update gradients of the memory cells, activation values, and time steps in the $t-1$ time step, as

$$\begin{aligned} dc^{t-1} &= \left(dc^t + da^t * \tau_o^t * \left[1 - \tanh^2\left(c^t\right)\right]\right) * \tau_f^t, \\ da^{t-1} &= W_{ca}{}^T * d\tilde{c}^t + W_{ua}{}^T \tau_u^t + W_{fu}{}^T \tau_f^t + W_{oa}{}^T \tau_o^t, \\ dx^t &= W_{cx}{}^T * d\tilde{c}^t + W_{ux}{}^T \tau_u^t + W_{fx}{}^T \tau_f^t + W_{ox}{}^T \tau_o^t, \end{aligned} \tag{4}$$

where W_c, W_u, W_f, and W_o are

$$W_c = \left[W_{ca} : W_{cx}\right], \quad W_u = \left[W_{ua} : W_{ux}\right], \quad W_f = \left[W_{fa} : W_{fx}\right], \quad W_o = \left[W_{oa} : W_{ox}\right]. \tag{5}$$

Derivatives of deviation values b_c, b_f, b_u, and b_o are shown as

$$db_c = \sum_{batch} d\tilde{c}^t \ , \quad db_f = \sum_{batch} d\tau_f^t, \quad db_u = \sum_{batch} \tau_u^t \ , \quad db_o = \sum_{batch} \tau_o^t. \tag{6}$$

2.2 CS algorithm

The cuckoo algorithm is a metaheuristic algorithm inspired by the brooding behavior of cuckoos and the flight patterns of leavers. It operates on the basis of three key rules.[15] (1) Cuckoos lay one egg at a time, placing it randomly in a nest; (2) nests with high-quality eggs are selected for the next generation; and (3) the number of host nests is fixed, and the probability of finding a cuckoo egg is $p_a \in (0,1)$. To simplify the process, it is assumed that each nest contains only one egg, and the numbers of eggs, nests, and cuckoos are the same. The algorithm includes both global exploration and local random walks. The local random walk is calculated using Eq. (7), where x_i^t and x_i^{t+1} represent the positions of the ith nest at iterations t and $t+1$, respectively.[16] Here, α is the step size scaling factor, s is the step size, s_0 is the minimum step size, ϵ is a

random number from a uniform distribution, and $\otimes$ denotes the dot product of vectors. Additionally, x_j^t and x_k^t represent two randomly selected positions at iteration t.

$$x_i^{t+1} = x_i^t + \alpha s \otimes H\left(p_a - \epsilon\right) \otimes \left(x_j^t - x_k^t\right), \quad H\left(x\right) = \begin{cases} 1, & x > 0 \\ 0, & x \le 0 \end{cases} \tag{7}$$

The Mantegna algorithm can achieve a symmetric Lévy stable distribution, and the step size s can be calculated via Eq. (8), where U follows a distribution with a mean of 0, the variance is a Gaussian normal distribution with x, and X can be calculated via Eq. (8). The τ function is constant for a given λ, and when $\lambda = 1$ and $\sigma^2 = 1$.

$$s = \frac{U}{\left|V\right|^{1/\lambda}}, \quad U \sim N\left(0, \sigma^2\right), V \sim N\left(0,1\right), \quad \sigma^2 = \left[\frac{\tau\left(1+\lambda\right)}{\lambda\tau\left(\left(1+\lambda\right)/2\right)} \bullet \frac{\sin\left(\pi\lambda/2\right)}{2^{(\lambda-1)/2}}\right]^{1/\lambda} \tag{8}$$

The global random walk is applicable to Lévy flights and is calculated using Eq. (9). λ is generally taken as 1 or 1.5, and α is taken as 0.01.

$$x_i^{t+1} = x_i^t + \alpha L\left(s, \lambda\right), \quad L\left(s, \lambda\right) = \frac{\lambda\tau\left(\lambda\right)\sin\left(\pi\lambda/2\right)}{\pi} \cdot \frac{1}{s^{(1+\lambda)}}. (s \gg s_0 > 0) \tag{9}$$

3. Methods

The BL model was used to detect DDoS attacks and was well-suited to scenarios with large datasets, multiple attributes, and interdependent time series. To enhance the performance of the BL network, in this paper, we proposed CSBLM, in which the CS algorithm is utilized to dynamically optimize key parameters, such as the number of hidden units in the LSTM layer and the time series length in the BL neural network. Through comparative experiments of the traditional GWO and PSO models, the effectiveness of CSBLM was demonstrated in terms of iteration times and optimization performance.

3.1 Neural network parameter optimization

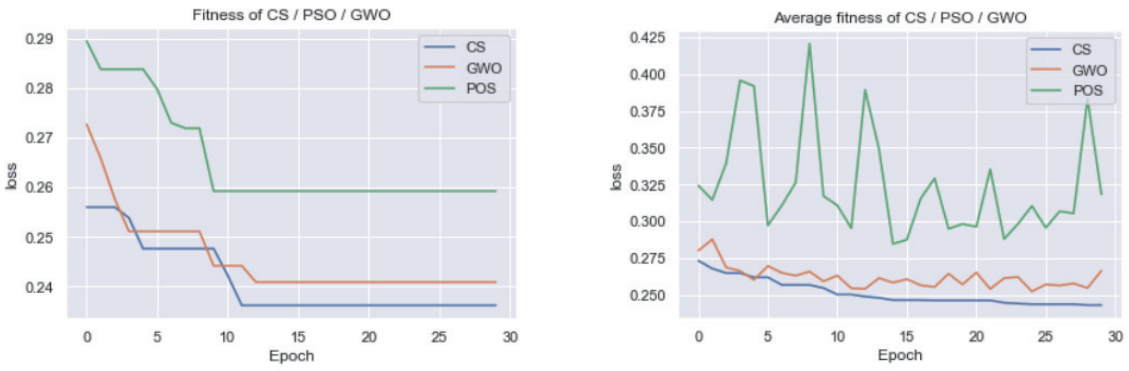
3.1.1 Comparison and analysis of CS, GWO, and PSO optimization algorithms

The BL network is effective for recognizing time series datasets because of its strong memory function. Its performance depends on both the dataset it is trained on and the network structure. Additionally, many parameters affect the network structure, and continuous experimentation is necessary to create a model that fits the specific dataset. With a large parameter space, random or exhaustive search methods require significant time and resources. CSBLM intelligently searches for the optimal combination of parameters in the network structure, such as time steps and the number of units in the LSTM layer, to create a more suitable

network for DDoS datasets and improve prediction accuracy. It is compared with models based on the GWO and PSO algorithms to demonstrate its efficiency. The GWO algorithm has strong convergence and fewer parameters, with each gray wolf representing a feasible solution moving towards the three best individuals in the cluster during optimization. The PSO algorithm relies on information sharing among individuals in the population, allowing the entire group to move towards the optimal solution by evolving from disorder to order.

In the experiment, the following parameters were set: number of nests, particles, and wolves $n = 8$, maximum number of iterations $iterationslter_{max} = 30$, and dimension $d = 2$. In the CS algorithm, a discarding factor $p_a = 0.25$ was applied. For the GWO algorithm, random values for r_1 and r_2 were generated within the range of 0 to 1, and the coordination coefficients A and C were calculated accordingly. In the PSO algorithm, the inertia parameter $w = 0.8$, the acceleration constants $C_1 = C_2 = 2$, and r_1 and r_2 were randomly generated within the range of 0 to 1. We used the IDS_ISCX_2012_dataset, which contains 100000 data points, with a 1:1 ratio of normal to malicious requests. The dataset was split into a 4:1 ratio for the training and test sets. As a result, 80000 data points were used for training the neural network, and the cross-entropy loss function value was calculated. This value served as the fitness function for the heuristic algorithm to determine the optimal parameter combination.

Figure 1 presents a comparison of the convergence effects in optimizing the BL model parameters using the CS, GWO, and PSO algorithms. The CS algorithm achieved optimal fitness values of 0.24766 in the fourth iteration and 0.23624 in the twelfth iteration. The GWO algorithm found better solutions in the fourth, tenth, and thirteenth iterations, with values of 0.25113, 0.24422, and 0.2409, respectively. The PSO algorithm reached a minimum fitness value of 0.28378 in the second iteration, and the value continued to decrease from the fourth to the tenth iteration, eventually stabilizing at 0.25925. In comparison, the cuckoo algorithm outperformed the other two methods in finding the optimal fitness value more quickly. This is because the CS algorithm performs both global and local optimizations in each iteration, with a higher probability of parameter replacement, leading to fewer iterations.



(a) (b)

Fig. 1. (Color online) (a) Convergence of the best-fit values of the three models and (b) average convergence values for three models.

The average fitness value of each iteration during the optimization process of the three algorithms is shown in Fig. 1(b). The GWO and CS algorithms show relatively smooth changes, while the PSO algorithm fluctuates between 0.28 and 0.425 as the number of iterations increases. This fluctuation is primarily due to PSO considering fewer factors when updating particle information, leading to a more random update speed than those in the other two algorithms. The CS algorithm is smoother than the GWO, as it updates position information by considering all individuals in the cluster, while GWO relies more on the positions of the three optimal gray wolves. As a result, the average fitness value of the CS algorithm consistently decreases with more iterations, while the GWO shows fluctuations during its decline, and the PSO shows more noticeable fluctuations. For detecting DDoS datasets, the cuckoo algorithm-based optimization model requires fewer iterations and converges more smoothly than both the gray wolf and particle swarm algorithms, demonstrating better overall performance.

3.1.2 Optimization of network parameters for the CSBLM

The CSBLM intelligently searches for the best combination of parameters in the BL neural network, including timesteps and LSTM layer units, to create a network more suited for DDoS datasets and improve prediction accuracy. To further validate the CS algorithm, we compare CSBLM with the GWO and PSO algorithms from earlier. The results show that the CS algorithm outperforms the others in optimization. The steps for optimizing the BL neural network parameters using the CS algorithm are as follows.

(1) Preprocess the IDS_ISCX_2012_dataset.
(2) Set the solution range and dimension of the BL neural network timesteps and units. Moreover, it is also necessary to set the number of nests in the heuristic algorithm, the probability of discarding P_a, and the maximum number of iterations.
(3) Initialize the variables and randomly assign the location of the nest in the CS algorithm. The location of the nest (timesteps, units) determines the subsequent composition of the BL network.
(4) Train the BL network model on the basis of the set parameters. Moreover, the cross-entropy error obtained after training is used as the fitness value of the heuristic algorithm, and later, it is also used as the criterion for whether to replace the old and new solutions.
(5) Global optimization is performed using the Levy formula to calculate the new nest location. The resulting solution is then placed into the BL network for training to obtain the corresponding fitness value. Finally, the fitness values of the new and old solutions are compared, and on the basis of the principle of optimal selection, a decision is made on whether to replace the old nest.
(6) After the global optimization step, the CS algorithm is applied for local optimization. During this process, nests with low fitness are discarded based on the basis of the probability P_a, and new nest positions are generated using the local random walk formula.
(7) Again derive the optimal solution on the basis of the fitness value, and then determine whether the maximum number of iterations has been reached. If not, return to step (6) to continue optimizing; otherwise, proceed to step (8).

(8) Obtain the optimal parameters of the BL network, establish the optimal network model, and finally detect DDoS data.

3.2 DDoS detection via CSBLM

As shown in Fig. 2, the CSBLM consists of two parts: parameter optimization and the neural network. The neural network is made up of an input layer, a hidden layer, and an output layer. The input layer normalizes the DDoS dataset and organizes it into time steps. The hidden layer uses LSTM units with a bidirectional structure to learn from both past and future time periods. Adjusting the number of neurons in the LSTM module enhances the network performance. The output layer, using a fully connected layer and sigmoid function, predicts the data types in the DDoS dataset. After the BL method, network parameters are updated through backpropagation with the help of optimization functions.

3.3 Experiment and results analysis

In this paper, we used the IDS_ISCX_2012 dataset and a three-layer network structure. By optimizing the network using CS, GWO, and PSO algorithms, the optimal combinations of time steps and the number of hidden units in the LSTM layer are found to be [51,19] and [62,22], respectively. The first value represents the time step, and the second represents the number of hidden units. Five metrics are selected for evaluation: accuracy, precision, recall, F1 score, and the number of iterations. Each parameter is tested 10 times, and the average value is calculated. Peng *et al.* used the parameter combinations [25,64] and achieved an accuracy of 97.89% and an F1 score of 97.89%.[(1)] These results are used for comparison in this study. During the experiment, the parameter values after the 8th and 40th iterations are recorded to observe the effect of iteration count on the results. Tables 1 to 6 present the data for three parameter combinations: [51,19], [25,64], and [62,22]. Tables 1 to 3 show the results after 8 iterations, while Tables 4 to 6

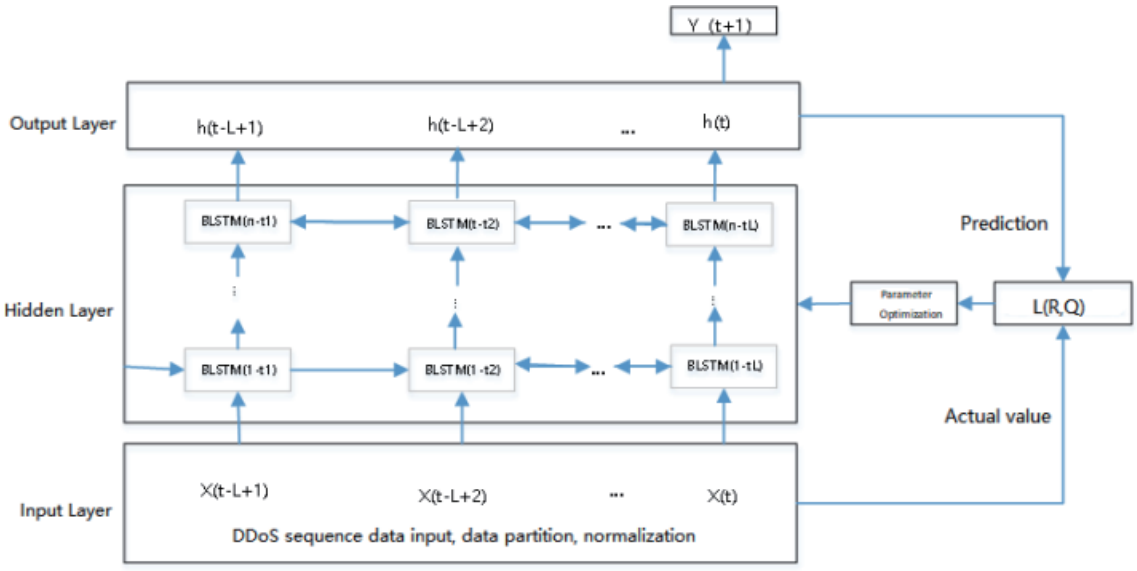


Fig. 2. (Color online) DDoS detection model based on the CSBLM network.

Table 1
Parameter combination [51,19], 8 iterations.

Test number	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)	Training loss (%)	Training acc (%)
1	93.62	95.53	91.68	93.75	0.1007	98.63
2	96.97	99.76	94.48	97.05	0.1088	98.35
3	95.70	99.53	92.43	95.85	0.1002	98.58
4	94.41	99.57	90.23	94.67	0.1031	98.42
5	91.18	94.77	88.38	91.47	0.1084	98.49
6	95.96	99.35	93.02	96.08	0.0912	98.84
7	95.38	97.58	93.45	95.47	0.1129	98.05
8	95.303	98.19	92.809	95.426	0.1085	98.46
9	95.168	98.596	92.252	95.319	0.1017	98.51
10	95.27	97.05	93.69	95.34	0.1008	98.60
Average value	94.90	97.99	92.24	95.04	0.104	98.49

Table 2
Parameter combination [25,64], 8 iterations.

Test number	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)	Training loss (%)	Training acc (%)
1	97.10	99.28	95.17	97.15	0.1323	96.71
2	96.97	99.76	94.48	97.05	0.1363	96.74
3	97.09	99.66	94.78	97.16	0.1345	96.68
4	96.72	99.83	93.97	96.82	0.1279	96.80
5	96.87	99.38	94.62	96.94	0.1372	96.54
6	97.03	99.41	94.89	97.10	0.1270	96.85
7	97.04	99.44	94.87	97.10	0.1339	96.77
8	97.02	99.74	94.59	97.10	0.1278	96.97
9	96.88	99.46	94.57	96.96	0.1290	96.69
10	96.89	99.72	94.37	96.97	0.1279	96.79
Average value	96.96	99.57	94.63	97.04	0.105	96.75

Table 3
Parameter combination [62,22], 8 iterations.

Times	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)	Training loss (%)	Training acc (%)
1	98.64	98.73	98.55	98.64	0.1132	98.03
2	99.11	99.78	98.46	99.12	0.1062	98.59
3	99.00	99.64	98.39	99.01	0.0993	98.64
4	98.57	99.19	97.98	98.85	0.1051	98.64
5	98.63	98.95	98.33	98.63	0.977	98.69
6	98.83	99.74	97.96	98.84	0.999	98.59
7	98.59	99.72	97.52	98.61	0.940	98.70
8	98.07	99.07	97.14	98.09	0.1008	98.70
9	99.15	99.83	98.5	99.16	0.1015	98.65
10	99.19	99.79	98.62	99.20	0.0910	98.86
Average value	98.78	99.44	98.15	98.82	0.363	98.61

Table 4
Parameter combination [51,19], 40 iterations.

Times	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)	Training loss (%)	Training acc (%)
1	99.60	99.93	99.29	99.61	0.0771	99.02
2	95.63	98.24	93.36	95.74	0.0833	98.78
3	93.83	99.42	89.40	94.15	0.0832	98.99
4	96.20	99.53	93.30	96.31	0.0803	99.05
5	95.66	96.96	94.49	95.71	0.0885	98.71
6	95.99	97.48	94.65	96.04	0.0847	98.81
7	93.63	91.05	95.98	93.45	0.0803	98.97
8	95.34	99.11	92.15	95.50	0.0948	98.72
9	94.98	99.73	91.06	95.20	0.0815	99.01
10	95.25	97.42	93.35	95.34	0.0911	98.82
Average value	95.61	97.89	93.70	95.71	0.084	98.89

Table 5
Parameter combination [62,22], 40 iterations.

Times	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)	Training loss (%)	Training acc (%)
1	98.99	99.79	98.21	99.00	0.881	98.79
2	99.25	99.90	98.60	99.25	0.875	99.04
3	99.33	99.73	98.94	99.33	0.075	99.14
4	99.10	99.67	98.53	99.10	0.0806	99.09
5	99.35	99.96	98.74	99.35	0.0715	99.25
6	99.13	99.89	98.39	99.14	0.0945	98.68
7	99.15	99.62	98.68	99.15	0.0782	99.11
8	99.17	99.88	98.48	99.18	0.773	99.13
9	98.17	99.15	97.23	98.18	0.0856	98.86
10	99.25	100	98.52	99.25	0.0767	99.07
Average value	99.09	99.76	98.43	99.09	0.309	99.02

Table 6
Parameter combination [25,64], 40 iterations.

Times	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)	Training loss (%)	Training acc (%)
1	98.37	99.99	96.89	98.41	0.1061	97.43
2	96.75	98.49	95.17	96.80	0.0975	97.68
3	97.88	99.34	96.53	97.91	0.1027	97.60
4	97.66	99.06	96.36	97.69	0.1001	97.67
5	97.46	99.74	95.38	97.51	0.1129	97.26
6	97.66	99.90	95.60	97.71	0.0993	97.69
7	97.80	99.37	96.33	97.83	0.1035	97.56
8	97.62	99.32	96.04	97.65	0.1033	97.67
9	97.98	99.07	96.65	98.00	0.1005	97.63
10	97.60	99.45	95.89	97.64	0.1011	97.77
Average value	97.68	99.37	96.08	97.72	0.103	97.60

display the results after 40 iterations. Each experiment was repeated 10 times, and the average values were calculated. The model was evaluated using the accuracy, precision, recall, and F1 score for the training set, and both the loss function value and accuracy of the training set were recorded.

Figures 3(a) and 3(b) display the accuracies for the three parameter combinations while Figs. 4(a) and 4(b) show their corresponding loss values. Notably, the experimental data are based on the training set and tested on the test set to check for overfitting. From the results in Figs. 4(a) and 4(b), it is concluded that the optimized parameter combinations achieve higher accuracy and lower loss values. Figure 3(a) shows that after 9 iterations, the parameter combinations [62,22] and [51,19] quickly reduce the loss rate and maintain an accuracy of about 98.5%, while the [25,64] combination has an accuracy of only around 96.5%. This trend

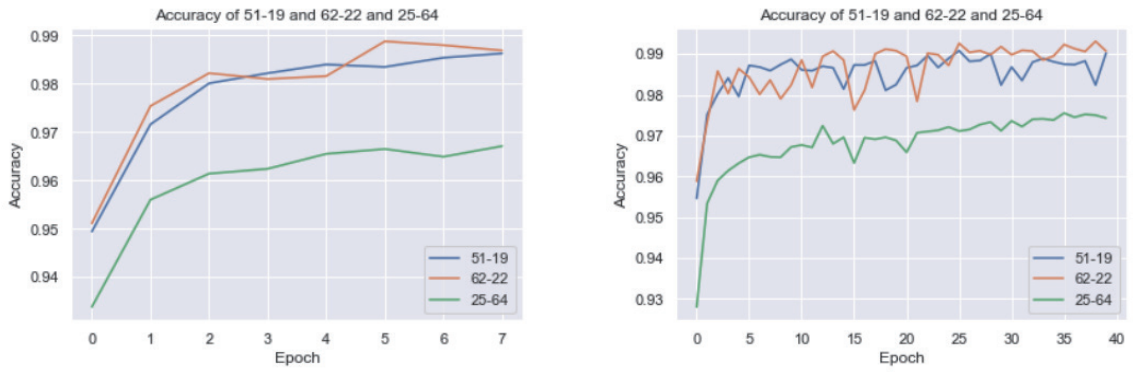


Fig. 3. (Color online) (a) Accuracy rates for three combinations after eight iterations. (b) Accuracies of the three combinations after 40 iterations.

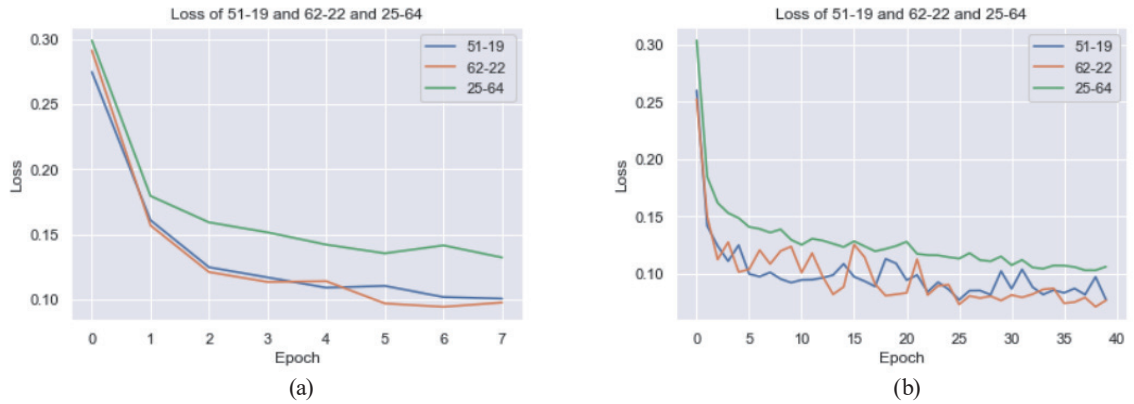
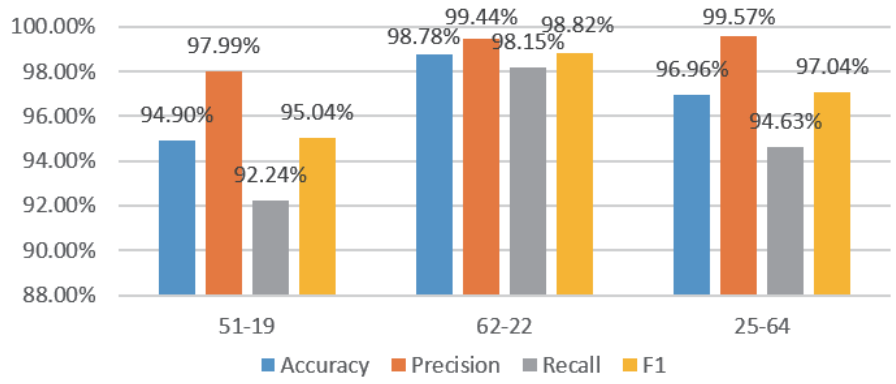


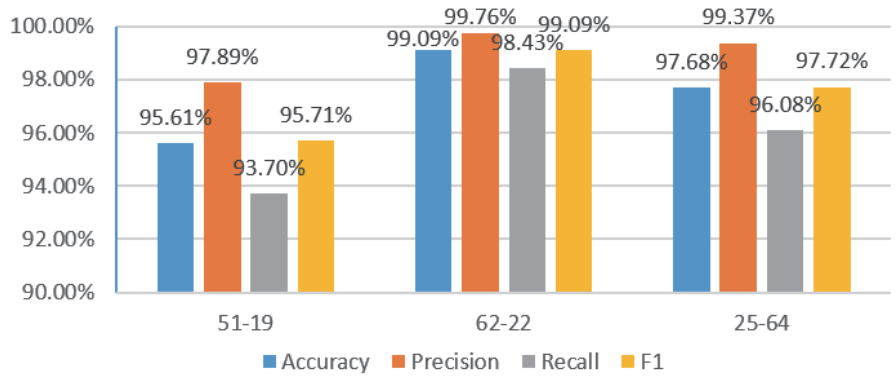
Fig. 4. (Color online) (a) Loss values for three combinations after eight iterations. (b) Loss values for 40 iterations of three combinations.

continues after 40 iterations. Figure 3(b) shows that the accuracies of [62,22] and [51,19] rise to approximately 99%, while that of [25,64] reaches only about 97.5%. Figures 4(a) and 4(b) show that the loss values for [62,22] and [51,19] decrease more quickly and remain lower for the same number of iterations. The loss value for [25,64] is significantly higher, indicating that parameter combinations optimized by heuristic algorithms can considerably improve both the accuracy and gradient descent speed in the training set.

Figures 5(a) and 5(b) show the results for three parameter combinations after 8 and 40 iterations. The data were averaged from 10 experiments and evaluated in terms of accuracy, precision, recall, and F1 score. According to the figures, for the 8 and 40 iterations, the accuracies of the [51,19] combination are 94.90% and 95.61%, respectively, which are lower than the approximately 98% accuracy seen in the training set. This indicates overfitting. While the parameter combinations [62,22] and [51,19] are optimized using heuristic algorithms, adjusting parameters based on fitness values alone does not prevent overfitting. To address this problem, optimization can be improved by increasing the number of iterations and recognition accuracy,



(a)



(b)

Fig. 5. (Color online) (a) Evaluation criteria for three sets of parameters after eight iterations. (b) Evaluation criteria for three sets of parameters after 40 iterations.

but additional measures are needed to prevent overfitting. One approach is to impose stricter constraints on the neuron optimization range to avoid overfitting caused by too few neurons in the LSTM layer or overly large time step divisions, while multiple optimization can help reduce the likelihood of such issues. For the [62,22] combination, the evaluation metrics for the training and testing sets are similar, and there is no clear evidence of overfitting. Compared with the [25,64] combination from previous literature,(1) accuracy increases from 96.96% and 97.68% after 8 and 40 iterations, respectively, to 98.78% and 99.09%.

4. Conclusions

We proposed an optimization model for the core parameters (time step size, number of neurons, etc.) of BL networks via CS. In terms of parameter selection, we compared and analyzed the performance of three heuristic algorithms, namely, CS, GWO, and PSO, in optimizing DDoS datasets and ultimately identified two optimal parameter combinations, [51,19] and [62,22]. These two combinations are compared with the parameter combination [25,64] reported in the literature.(13) The experimental results show that the two combinations [51,19] and [62,22] have better performance for the training set and can achieve higher accuracy in fewer iterations. Moreover, a comparison of the performance of the three methods on the test set reveals that the combination method of [51,19] has an overfitting phenomenon. We adopted a more precise setting of the neuron range and multiple optimization testing methods to solve the overfitting problem. With the parameter combination of [62,22], the accuracies can reach 98.78% and 99.09% in the 8th iteration and 40th iteration, respectively, and the F1 value also reaches 98.81% and 99.09%, which are considerable improvements.

Acknowledgments

This work is supported in part by the Fujian Provincial Department of Education Project (No. JAT241409), Fujian Provincial Science and Technology Special Commissioner Project (No. 202135060173), and Zhangzhou Science and Technology Special Commissioner Project (No. 2024S350623028).

References

1 T. Peng, C. Leckie, and K. Ramamohanarao: ACM Comput. Surv. **39** (2007) 3–es.
2 G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and M. Rajarajan: Comput. Electr. Eng. **59** (2017) 165.
3 https://www.wired.com/story/github-ddos-memcached/ (accessed March 2018).
4 https://www.zdnet.com/article/aws-said-it-mitigated-a-2-3-tbps-ddos-attack-the-largest-ever/ (accessed June 2020).
5 A. Bhardwaj, V. Mangat, and R. Vig: IEEE Access **8** (2020) 181916.
6 R. K. C. Chang: IEEE Commun. Mag. **40** (2002) 42.
7 T. Mahjabin, Y. Xiao, G. Sun, and W. Jiang: Int. J. Distrib. Sens. Netw. **12** (2017) 13.
8 https://www.techtarget.com/searchnetworking/definition/TCP (accessed March 2024).
9 U. Rahamathullah and E. Karthikeyan: Proc. Int. Conf. Smart Data Intelligence (ICSMDI 2021), Tamil Nadu, India.
10 Z. Li, H. Jin, D. Zou, and B. Yuan: IEEE Trans. Parallel Distrib. Syst. **31** (2020) 695.

11 T. Shon, Y. Kim, C. Lee, and J. Moon: Proc. 6th Annu. IEEE SMC Information Assurance Workshop, West Point, NY, USA, 2005, 176–183.
12 X. Yuan, C. Li, and X. Li: 2017 IEEE Int. Conf. Smart Computing (SMARTCOMP), Hong Kong, China, 2017, 1–8.
13 L. Jin, J. Zhang, Z. Li, and Y. Liao: 2024 5th Int. Conf. Big Data & Artificial Intelligence & Software Engineering (ICBASE), Wenzhou, China, 2024, 579–583.
14 J. Schmidhuber and S. Hochreiter: Neural Comput. **9** (1997) 1735.
15 Y. Zhang and Z. Jin: J. Harbin Inst. Technol. **51** (2019) 89.
16 Y. Li, Z. Shang, and J. Liu: J. Comput. Sci. **47** (2020) 219.