electronics

MDPI

Article

Deep Learning Short Text Sentiment Analysis Based on Improved Particle Swarm Optimization

Yaowei Yue [1,*], Yun Peng [1,2,*] and Duancheng Wang [1]

[1] School of Computer and Information Engineering, JiangXi Normal University, Nanchang 330224, China; wangdc@jxnu.edu.cn
[2] School of Digital Industry, JiangXi Normal University, Shangrao 334000, China
* Correspondence: 202040100739@jxnu.edu.cn (Y.Y.); pengyun@jxnu.edu.cn (Y.P.)

Abstract: Manually tuning the hyperparameters of a deep learning model is not only a time-consuming and labor-intensive process, but it can also easily lead to issues like overfitting or underfitting, hindering the model's full convergence. To address this challenge, we present a BiLSTM-TCSA model (BiLSTM combine TextCNN and Self-Attention) for deep learning-based sentiment analysis of short texts, utilizing an improved particle swarm optimization (IPSO). This approach mimics the global random search behavior observed in bird foraging, allowing for adaptive optimization of model hyperparameters. In this methodology, an initial step involves employing a Generative Adversarial Network (GAN) mechanism to generate a substantial corpus of perturbed text, augmenting the model's resilience to disturbances. Subsequently, global semantic insights are extracted through Bidirectional Long Short Term Memory networks (BiLSTM) processing. Leveraging Convolutional Neural Networks for Text (TextCNN) with diverse convolution kernel sizes enables the extraction of localized features, which are then concatenated to construct multi-scale feature vectors. Concluding the process, feature vector refinement and the classification task are accomplished through the integration of Self-Attention and Softmax layers. Empirical results underscore the effectiveness of the proposed approach in sentiment analysis tasks involving succinct texts containing limited information. Across four distinct datasets, our method attains impressive accuracy rates of 91.38%, 91.74%, 85.49%, and 94.59%, respectively. This performance constitutes a notable advancement when compared against conventional deep learning models and baseline approaches.

Keywords: sentiment analysis; particle swarm optimization; GAN; convolution neural network; self-attention mechanism

Citation: Yue, Y.; Peng, Y.; Wang, D. Deep Learning Short Text Sentiment Analysis Based on Improved Particle Swarm Optimization. *Electronics* **2023**, *12*, 4119. https://doi.org/10.3390/electronics12194119

Academic Editor: Heung-Il Suk

Received: 20 August 2023
Revised: 23 September 2023
Accepted: 28 September 2023
Published: 2 October 2023

1. Introduction

Text sentiment analysis typically involves the exploration and examination of the emotional inclination present within textual commentaries. This encompasses the extraction of subjectively imbued elements like viewpoints, stances, attitudes, and opinions, subsequently entailing the assessment of the emotional underpinnings of said elements. The proliferation of the internet and the rapid evolution of social media platforms and online comment forums have propelled text sentiment analysis into a realm of scholarly inquiry that commands considerable focus in recent years.

Currently, proposed methods for text sentiment analysis can be broadly categorized into two main groups. The first category entails machine learning approaches, encompassing techniques like support vector machines [1], K-nearest neighbor classification algorithms [2], naive Bayes [3], and Latent Dirchlet allocation [4]. While these algorithms exhibit commendable performance, they predominantly hinge upon manually extracting text features, leading to inefficiencies throughout the entire process. The second approach centers around deep learning, a prevailing paradigm within the realm of natural language processing. Prominent examples encompass the Convolutional Neural Networks (CNN)

and the Recurrent Neural Networks (RNN). The former, CNN, adeptly discerns local features from textual matrices through convolutional and pooling operations. Notably, Kim [5] introduced a CNN-based sentence classification model that yields promising classification results across multiple public datasets. However, it struggles to effectively to grasp global context features. Recognizing the shortcomings in obtaining comprehensive textual semantics and contextual insights, sources in the literature [6–8] increase the number of layers to increase the receptive field of information obtained by the convolutional kernel, but the effect is not significant. For this reason, many researchers began to change the structure of the convolution layer and the pooling layer from the horizontal. Guo [9] ingeniously constructed an augmented CNN model, harnessing skip convolution and K-max pooling operations, thereby refining the extraction of features from concise texts. Additionally, Wang [10] fortified the TextCNN architecture through two distinct avenues: the creation of N-gram discontinuous sliding windows and K-Max average pooling. Despite these advancements, owing to the sparse nature of textual content and the inherent limitations in capturing temporal dimensions, even these enhanced models grapple with inadequacies in fully acquiring the nuances of text semantics.

RNN possesses the capability to extract global information to a certain extent. Irsoy [11] harnessed RNN for extracting sequence features in sentiment analysis tasks. Similarly, Xin [12] employed a specialized RNN structure to capture text features for subsequent analysis. However, due to its inherent limitations in "memory" capacity, RNN encounters issues such as gradient explosion or vanishing gradients when text length exceeds a certain range, thereby impacting experimental outcomes. In response to this challenge, Hochreiter and Gers [13,14] introduced the LSTM network, and in subsequent advancements, Cho [15] amalgamated input and forget gates into update gates, culminating in the development of Gated Recurrent Units (GRUs). This not only effectively resolves the gradient vanishing or explosion problem but also refines the internal processing units, enabling more adept context information capture. Consequently, GRUs outperform RNNs on numerous tasks. Tai [16] introduced textual topological structure information into LSTM models, constructing LSTM models based on both dependency trees and phrase structure trees. This novel approach modifies the network structure to make gate vectors and internal state updates dependent on states from multiple relevant subunits, effectively amalgamating syntactic features such as dependency relationships and phrase composition in short texts. This results in more accurate semantic representations for short texts. Zhao's [17] proposed sequential hybrid model initially employs BiLSTM to capture the global features of the text and subsequently utilizes CNN to extract local semantic features and perform classification. This approach mitigates the loss of positional information and contributes to enhanced results.

In recent years, attention mechanisms [18] have been introduced to assign significant attention weights to vectors, highlighting key information. The integration of attention mechanisms enhances feature expression, resulting in models with higher accuracy that intuitively emphasize the importance of different words. Yang [19] combined a hierarchical attention network with Bidirectional Gated Recurrent Units (Bi-GRU) for English short text classification. This approach segments text into three hierarchical levels—words, sentences, and documents—introducing an attention mechanism between each level to assign varying weights to different words and sentences, progressively selecting crucial information. Building upon Yang's work, Zhou [20] proposed a hybrid attention network that combines both word-level and character-level information for classifying Chinese short texts. By merging the attention mechanism with BiLSTM, this model captures both word and character embeddings, thus extracting the most pivotal word and character information. Cheng Yan's model [21] integrates a multi-channel CNN with Bi-GRU. This combination exploits the complementarity of local features and contextual information, with the incorporation of an attention mechanism further identifying pivotal information within the text.

While the aforementioned methods have yielded promising results, the configuration of hyperparameters significantly influences the performance of deep learning predictive models. Relying on manual parameter tuning based on empirical knowledge introduces considerable errors, thereby limiting the model's full potential. To address this issue, Feurer [22] introduced an advanced interface capable of automatically assessing multiple preprocessing and model fitting pipelines. They delved into the concept of achieving automated machine learning (AutoML) through meta-learning, with the aim of streamlining the tedious manual tuning process involved in hyperparameter optimization for deep learning models. Wang [23] proposed an efficient and robust method for automated hyperparameter tuning using Gaussian processes. This approach effectively handles various hyperparameter combinations and employs Gaussian process regression to predict model performance. However, Jeremy [24] breaks away from tradition through a method known as Automatic Gradient Descent (AGD), which allows for training deep learning models without the need for hyperparameters.

This paper presents an innovative approach that integrates an IPSO into a deep learning-based sentiment analysis model for text. Leveraging BiLSTM, TextCNN, and the Attention Mechanism, this model adeptly extracts profound semantic information from text pairs. Furthermore, it introduces a Generative Adversarial Network (GAN) mechanism to generate perturbed texts for training and combines with the IPSO to optimize and match the model's hyperparameters effectively. Experimental results further underscore the efficacy and feasibility of this approach.

2. Related Works

2.1. IPSO

In general, deep learning models often have numerous hyperparameters, and their search space can be extensive. When employing manual search methods or grid search techniques, it can become impractical due to the presence of continuous hyperparameter variables. This not only makes the search process cumbersome but also incurs high computational time costs. Assuming that a three-dimensional function '$Z = g(x,\ y)$' needs to find its optimal integer solution, when the search range of 'x' is denoted as 'm' and the search range of 'y' is denoted as 'n', as the search ranges 'm' and 'n' continue to expand, the scope of grid search grows quadratically, the horizontal and vertical axes represent the expansion of the search domains, 'm' and 'n', while the vertical axis represents the number of parameter tuning iterations, as illustrated in Figure 1. In this, areas with a redder color indicate a higher number of required iterations.

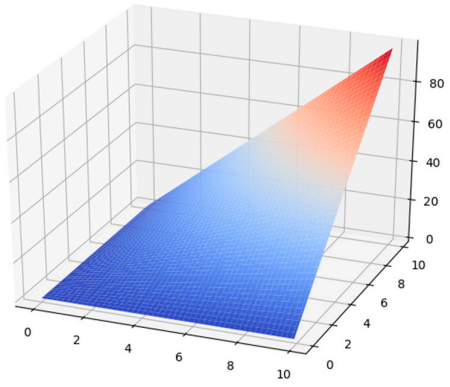


Figure 1. Parameter tuning iterations in grid search.

To address the challenge of manually exhaustively searching for optimization solutions, various specialized algorithms have been proposed. These include the Pathfinder Algorithm (PA) [25], Multi-Verse Optimizer (MVO) [26], Moth Flame Optimizer (MFO) [27], and Arithmetic Optimization Algorithm (AOA) [28].

The PA algorithm draws inspiration from the behavior of ants searching for the shortest path. It employs communication based on pheromones to guide the search for the optimal path in the problem space. However, its applicability is limited when it comes to problems unrelated to pathfinding or routing.

The MVO algorithm is based on the concept of multiple universes, where each universe represents a set of solutions. These universes interact, share information, and compete to find the best solutions, making it suitable for multimodal optimization problems. However, when a large number of populations are set, the computational resources required increase significantly, which may not be acceptable.

The MFO algorithm is inspired by the behavior of moths being attracted to flames. Moths represent solutions, and they are attracted to better solutions while avoiding the "flames" or undesirable regions. This algorithm iteratively updates solutions to find the best or near-optimal solution. It is easy to implement and applicable to various optimization problems, but it can easily become stuck in local optima, especially in complex high-dimensional spaces.

The AOA algorithm is a general optimization framework based on arithmetic operations such as addition, subtraction, multiplication, and division. It iteratively improves solutions based on the solutions themselves and has a wide range of applications. However, for complex problems, it may require a substantial amount of computational resources.

In comparison to those algorithms mentioned above, the Particle Swarm Optimization (PSO) algorithm boasts several advantages. It exhibits minimal dependence on initial information, avoids convergence into local minima due to high computational complexity, and is relatively straightforward to comprehend. Originally introduced by Kennedy and Eberhart [29], PSO is a global random search algorithm inspired by the foraging behavior of birds. In complex constrained problems, particles represent optimal solutions under multiple optimization constraints, and a particle swarm constitutes a collection of these optimal solutions. The dimensionality of particles is determined by the number of parameters to be optimized, while the number of particles in the swarm can be adjusted based on factors such as the quantity of optimization problems, required computational resources, and optimization objectives.

The algorithm sets the position state 'X_i' and velocity state 'V_i' to represent the current state of the particle 'i'. Throughout the iterative process, particles continually update their positions and velocities based on the interplay between individual and global best values of the current state. Upon locating the present global and individual optimal solutions, particle information is iteratively updated according to the following Equations (1)–(4) [30,31]:

$$X_i = (x_{i1}, x_{i2}, x_{i3}, \ldots, x_{in}) \tag{1}$$

$$V_i = (v_{i1}, v_{i2}, v_{i3}, \ldots, v_{in}) \tag{2}$$

$$V_{id}^{t+1} = wv_{id}^{t} + c_1 rand_1(pbest_{id}^{t} - x_{id}^{t}) + c_2 rand_2(gbest_{d}^{t} - x_{id}^{t}) \tag{3}$$

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{4}$$

Among them, 'X_i' and 'V_i' represent the collection of positions and velocities of particles in each dimension of the current n-dimensional space. The most crucial step in the particle swarm optimization algorithm lies in Equation (3), which is used to update the velocity information of the current particle in a specific dimension. Here, 'w' denotes the weight of the current velocity, 'c_1' and 'c_2' represent learning factors, '$rand_1$' and '$rand_2$' are random numbers between (0, 1), and '$pbest$' and '$gbest$' stand for the individual best value found by the current particle and the global best value found by all particles, respectively. Therefore, in Equation (3), the velocity increment information 'V_{t+1}' depends not only on the current position 'V_t' but also on the individual best '$pbest$' of the particle and the global

best solution '*gbest*' found by all particles. Finally, the current position information of the particle is updated using Equation (4).

To validate the superior performance of the PSO algorithm, this paper compares the PSO algorithm with other algorithms such as PA, MVO, MFA, and AOA using the CEC benchmark test function set. The mathematical formulas for the CEC standard benchmark functions are presented in Table 1, and the results of various algorithms on the CEC standard benchmark functions are presented in Table 2.

Table 1. CEC standard benchmark functions.

Function Name	Mathematical Expression
axis_parallel_hyper_ellipsoid	$f(x) = \sum_{i=1}^{D} x_i^2 \times 10^6$
rosenbrocks_valley	$f(x) = \sum_{i=1}^{D-1} 100 \times (x_{i+1} - x_i^2)^2 + (1 - x_i)^2$
schwefel	$f(x) = -\sum_{i=1}^{D} x_i \times \sin(\sqrt{\lvert x_i \rvert})$
ackley	$f(x) = -20 \times \exp\left[-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{D} x_i^2}\right] - \exp\left[\frac{1}{n}\sum_{i=1}^{D} \cos(2\pi x_i)\right] + 20 + e$
Griewank	$f(x) = \frac{1}{4000} \times \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} (\cos(\frac{x_i}{sqrt(i)})) + 1$

Table 2. The final convergence values of various algorithms on the CEC standard benchmark functions.

	Axis Parallel Hyper Ellipsoid	Rosenbrocks Valley	Schwefel	Ackley	Griewank
PA	$\mathbf{1.28\times10^{-7}}$	$\mathbf{1.19\times10^{-7}}$	830.983	$\mathbf{3.62\times10^{-4}}$	$\mathbf{7.09\times10^{-5}}$
MVO	$\mathbf{1.43\times10^{-7}}$	$\mathbf{5.82\times10^{-8}}$	830.983	$\mathbf{1.03\times10^{-4}}$	$\mathbf{1.29\times10^{-6}}$
MFA	$\mathbf{1.99\times10^{-8}}$	$\mathbf{6.22\times10^{-8}}$	830.075	$\mathbf{1.13\times10^{-4}}$	$\mathbf{9.86\times10^{-7}}$
AOA	$\mathbf{2.21\times10^{-3}}$	$\mathbf{6.13\times10^{-3}}$	830.075	$\mathbf{1.26\times10^{-1}}$	$\mathbf{5.61\times10^{-3}}$
PSO	$\mathbf{1.75\times10^{-8}}$	$\mathbf{1.07\times10^{-9}}$	**830.069**	$\mathbf{5.51\times10^{-5}}$	$\mathbf{9.52\times10^{-8}}$

While PSO exhibits notable performance in optimization compared to other algorithms, this paper aims to further enhance the performance of the PSO. Specifically, we propose improvements to the inertia 'w' and learning factors 'c_1' and 'c_2' within the PSO.

2.1.1. Improvement of the Inertia Weight 'w'

The inertia weight refers to the extent to which the current velocity is influenced by the previous generation's velocity. It is directly proportional to the global search capability and inversely proportional to the local search capability. During the early iterations, the particle's optimal state lies in possessing a strong global search capability, enabling it to explore new regions and discover potential possibilities. However, as the iterations progress, there arises a need to enhance the particle's local search capability to ensure smooth convergence. Therefore, a novel approach is introduced to dynamically adjust the inertia weight 'w' using a nonlinear function. The formula is as follows in Equation (5):

$$w = w_{\max} - (w_{\max} - w_{\min}) \times \frac{4}{\pi} \times \arctan\frac{t}{t_{\max}} \tag{5}$$

The motivation for using '$\frac{4}{\pi} \times \arctan\frac{t}{t_{max}}$' is twofold: it exhibits the characteristics of a convex function and can effectively adapt to the iterative process of the weight 'w' operation. In the early stages of iteration, although individual diversity is required to explore unknown domains, the portion controlled by the inertia weight does not dominate. Therefore, it is essential to apply a certain degree of decay rate to the 'w' in the early

stages, maintaining a relatively high level of acceleration in the 'w' decay. However, as the iteration progresses, in order to expedite particle convergence to the global optimum, it becomes necessary to smoothly reduce the 'w' to its minimum value. Consequently, in the later stages, the acceleration of w decay should be maintained at a lower level. Assuming '$w_{max} = 0.8$' and '$w_{min} = 0.2$', the variation of the 'w' decay acceleration during the iteration process is illustrated in Figure 2.

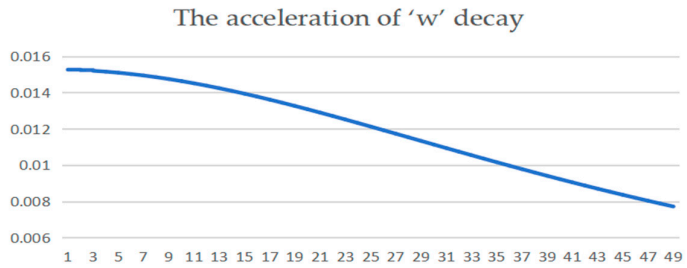


Figure 2. The acceleration of 'w' decay in iterations.

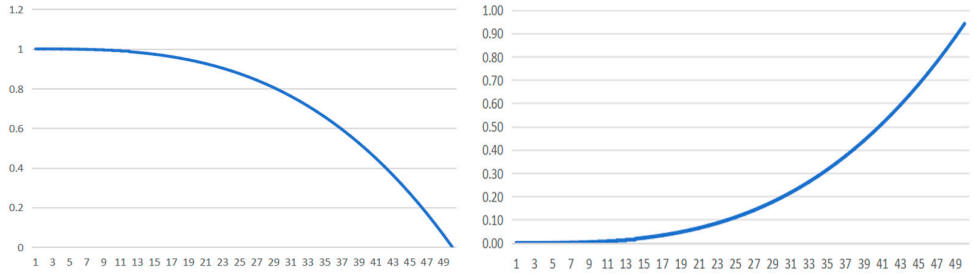
2.1.2. Improvement of Learning Factors

The learning factors, 'c_1' and 'c_2', respectively determine the extent to which particles are influenced by group information and individual information. When 'c_1' is set to 0, the particle's state is entirely governed by global information, resulting in reduced diversity within the group and making it difficult for particles to escape local optima. Conversely, when 'c_2' is set to 0, the particle's state solely relies on individual information, displaying a self-centric behavior that disregards the contributions from the group. This can lead to a slower convergence rate of the algorithm. Hence, a nonlinear function is introduced to dynamically adjust the learning factors, following Equations (6) and (7):

$$c_1(t) = (c_{1e} - c_{1s})(t/t_{\max})^3 + c_{1s} \quad (6)$$

$$c_2(t) = (c_{2e} - c_{2s})(t/t_{\max})^3 + c_{2s} \quad (7)$$

where 'c_{1s}' and 'c_{1e}' represent the initial and final values of parameter 'c_1', with '$c_{1s} = 1$' being greater than '$c_{1e} = 0$', and 'c_{2s}' and 'c_{2e}' denoting the initial and final values of parameter 'c_2' with '$c_{2s} = 0$' being less than '$c_{2e} = 1$'. From the perspective of 'c_1' during the iteration process, it is necessary to slowly decay 'c_1' using the concave function characteristic of '$\left(\frac{t}{t_{max}}\right)^3$'. This is because, in the early stages of iterations, particles need to possess sufficient individual skills to explore unknown territories. Therefore, the rate of 'c_1' decay in the early stages should be slow, and the gradient should decrease. However, in the later stages, when particles no longer need to explore unknown areas due to their independent characteristics being suppressed or even cleared, the acceleration of 'c_1' decay should increase. During iterations, the variation of 'c_1' is illustrated in Figure 3a. On the contrary, for 'c_2', in the early stages of iteration, it is necessary to suppress the global characteristics of particles, allowing them to move with their "self-will". Therefore, only a slight increase in 'c_2' is required in the early stages. However, as the iterations progress to the later stages, the global optimum requires a significant "attraction" to particles. Thus, 'c_2' should undergo a substantial increment. The concave function characteristic of '$\left(\frac{t}{t_{max}}\right)^3$' can meet the requirements for 'c_2"s incremental changes during the iteration. Based on the description of 'c_2', the variation of 'c_2' will be as shown in Figure 3b.



(**a**) (**b**)

Figure 3. The variation of 'c_1' and 'c_2' in iterations. (**a**) The variation of 'c_1'. (**b**) The variation of 'c_2'.

The specific computation process of the IPSO algorithm is illustrated in Figure 4. The algorithm consists of the following six specific steps:

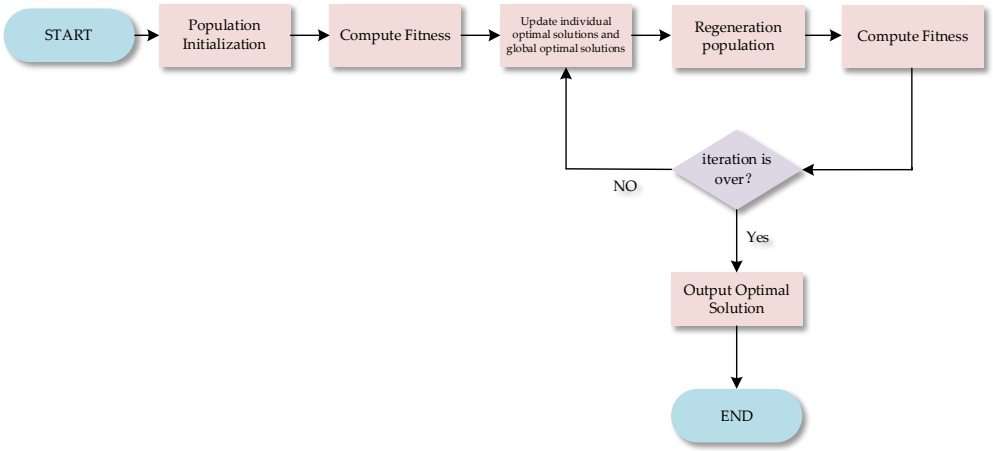


Figure 4. IPSO Algorithm Flow.

(1) Randomly generate an initial batch of particles (along with initialized particle velocities and positions) to constitute the current population.
(2) Calculate the fitness for each particle.
(3) Update the individual best and global best solutions for each particle based on their fitness function values. The updated equations are listed as Equation (8). Here, 'p_i' represents the fitness value of the particle at the current iteration 'i', 'g_i' represents the global best solution at iteration 'i', and 'n' is the maximum number of iterations.

$$pbest = \min(p_1, p_2, p_3, \ldots, p_n) \tag{8}$$

$$gbest = \min(g_1, g_2, g_3, \ldots, g_n) \tag{9}$$

(4) Updating particle velocities and positions using Equations (3) and (4).
(5) Calculate the fitness for each particle again.
(6) Check if the iteration is complete. If it is, output the current best solution. Otherwise, proceed to (3).

2.1.3. Performance Analysis of the IPSO

To evaluate the performance of the IPSO, this study utilizes three CEC standard benchmark functions to demonstrate the algorithm's improvements, namely, '*holder_table*', '*easom*', and '*rastrigin*', with their expressions shown in Equations (10)–(12). Since it

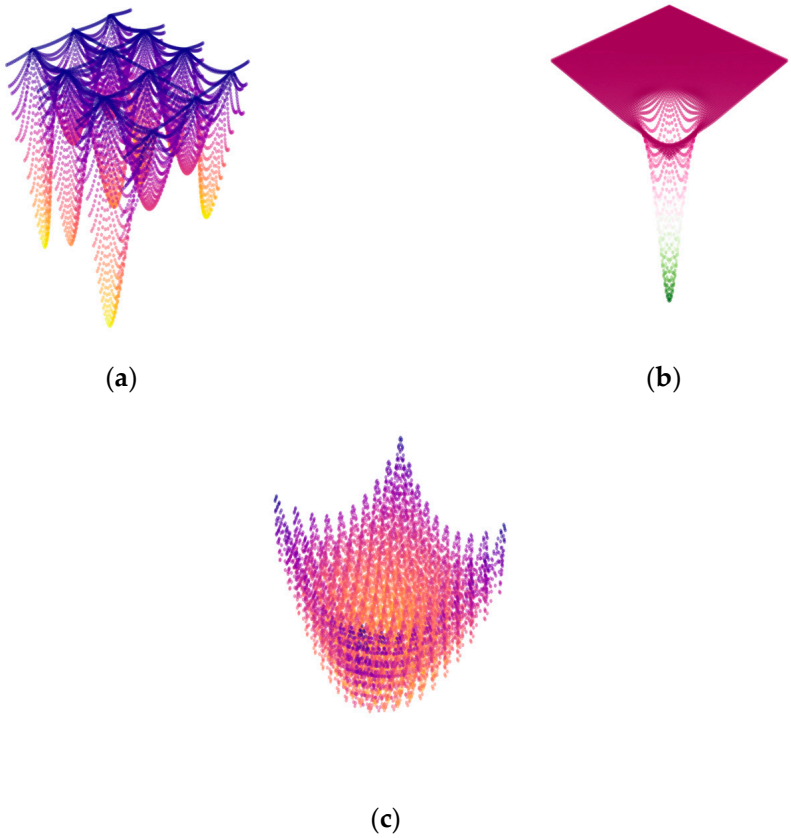
involves finding the global optimum of multivariate functions, the fitness function in the IPSO is the test function itself.

$$holder\_table(x) = -\left|\prod_{i=1}^{D}\sin(x_i)\cos(x_i) \times \exp\left(\frac{1-\sqrt{\sum_{i=1}^{D}x_i^2}}{\pi}\right)\right| \tag{10}$$

$$easom(x) = -\prod_{i=1}^{D}[\cos(x_i)] \times \exp\left[\sum_{i=1}^{D}(x_i-\pi)^2\right] \tag{11}$$

$$rastrigin(x) = 10n + \sum_{i=1}^{D}\left[x_i^2 - 10\cos(2\pi x_i)\right] \tag{12}$$

Figure 5 illustrates the three-dimensional plot of the test function '*holder_table*', '*easom*', '*rastrigin*'.



(**a**) (**b**)

(**c**)

Figure 5. Three-Dimensional Plot of the (**a**) '*holder_table*', (**b**) '*easom*', and (**c**) '*rastrigin*'.

During testing, a particle swarm size of 20 and particle dimensionality of 2 were configured, with a maximum iteration count of 200. The parameters were set as $'w_{max}' = 0.8$ and $'w_{min}' = 0.4$. A comparative analysis was conducted on the evolution of fitness curves between the pre-improvement and post-improvement methods, as depicted in Figure 6.

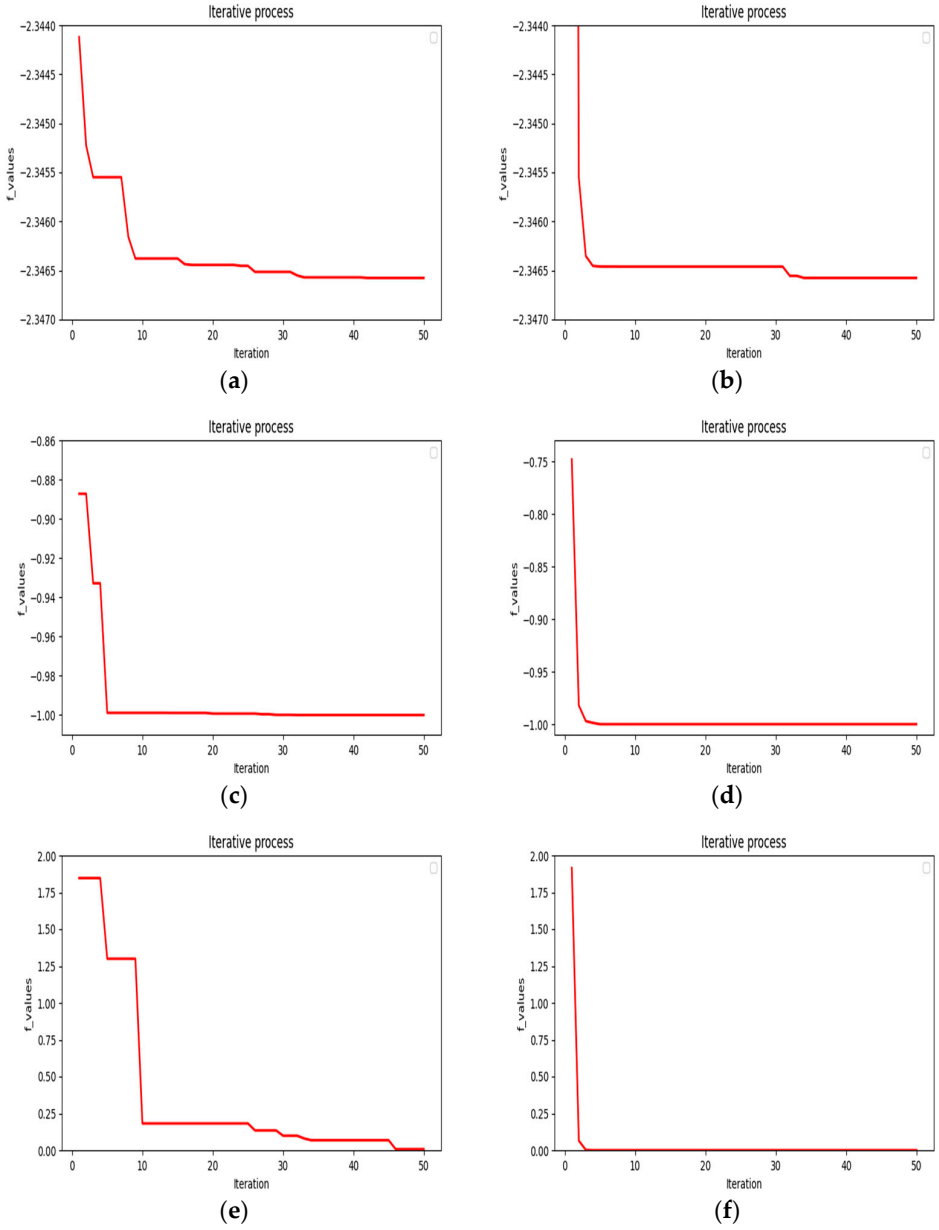


Figure 6. Analysis of the PSO and IPSO in '*holder* − *table*', '*easom*' and '*rastrigin*'. (**a**) PSO in '*holder* − *table*'. (**b**) IPSO in '*holder_table*'. (**c**) PSO in '*easom*'. (**d**) IPSO in '*easom*'. (**e**) PSO in '*rastrigin*'. (**f**) IPSO in '*rastrigin*'.

From the comparison in the above figures, it can be observed that the performance of PSO is not optimal, and it exhibits certain shortcomings compared to IPSO. Comparing Figure 6a with Figure 6b, as well as Figure 6c with Figure 6d, it can be seen that in the early iterations, PSO lacks sufficient individual diversity, leading to insufficient capability to escape local optima and often becoming trapped in local optima briefly and frequently. By examining Figure 6e,f, it becomes evident that the global characteristics of particles in PSO do not transition with iterations, resulting in unstable update speeds. In contrast, particles in IPSO exhibit significant individual diversity in the early stages, allowing them to thoroughly explore the unknown space and converge to the global optimum solution quickly in the initial iterations. Furthermore, as the iterations progress, the particles in IPSO quickly stabilize.

2.2. BiLSTM

In the LSTM network, although the model circumvents the issues of vanishing or exploding gradients seen in RNN, its unidirectional nature restricts it to utilizing information from the 'preamble', leaving it unaware of the 'postamble'. In practical scenarios, predictive outcomes often necessitate access to information from the entire input sequence. Consequently, the prevalent approach involves the application of BiLSTM [32]. Building upon the foundation of LSTM, BiLSTM integrates input sequence information from both the forward and backward directions. For the output at time 't', the forward LSTM layer incorporates information up to and including time 't', while the backward LSTM layer incorporates information from 't' onwards in the input sequence. The outputs from both forward and backward layers are then averaged to obtain the BiLSTM layer's output. The architecture of the BiLSTM network is illustrated in Figure 7.

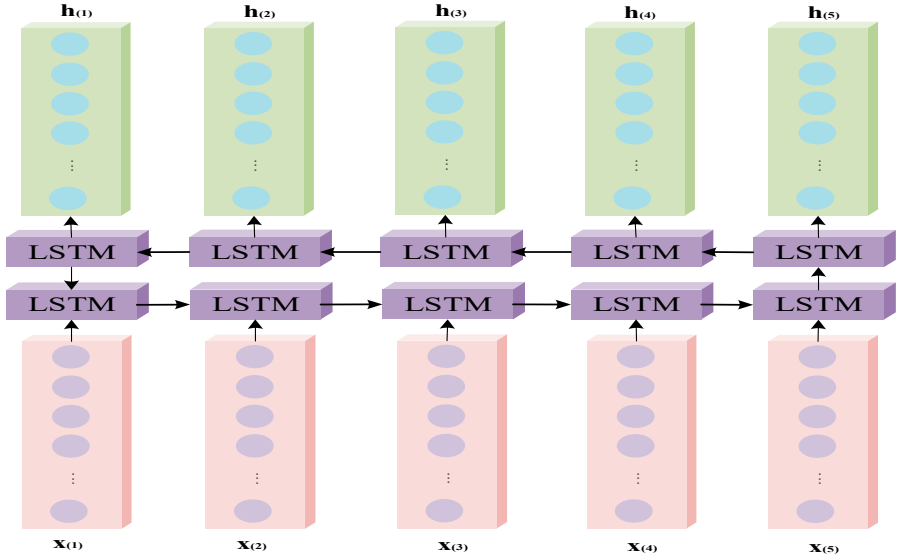


Figure 7. The Structure of BiLSTM.

The gating mechanisms within the hidden layers of the LSTM structure determine the transmission of information and have the capacity to learn critical dependencies for the current information. The forget gate governs the discarding of information that is deemed unimportant for classification, the input gate determines which information needs to be updated, and the output gate determines the information to be output. Let us denote the values of the forget gate, input gate, and output gate at time 't' as 'f_t', 'i_t', and 'o_t', respectively. Specifically, the activation function 'δ' for the forget gate utilizes the sigmoid function. The information update is represented as follows in Equation (13):

$$f_t = \delta(W_f \times X + b) \tag{13}$$

Information update for the input gate is computed using Equation (14):

$$i_t = \delta(W_i \times X + b_i) \tag{14}$$

Computation for updating the output gate's information is given by Equation (15):

$$O_t = \delta(W_o \times X + b) \tag{15}$$

Calculation for updating the cell information state is represented by Equation (16):

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c \times X + b_c) \tag{16}$$

The update of the hidden state information at time 't' is determined by Equation (17):

$$h^{(t)} = O_t \odot \tanh(c_t) \tag{17}$$

where '$h^{(t)}$' is composed of the forward '$h^{(t)}$' and backward '$h^{(t)}$' states, and 'W_f', 'W_i', 'W_o', and 'W_c' represent the weight matrices for the forget gate, input gate, output gate, and updated cell state, respectively. Correspondingly, 'b_f', 'b_i', 'b_o', and 'b_c' are the respective bias terms. The LSTM cell structure is illustrated in Figure 8.

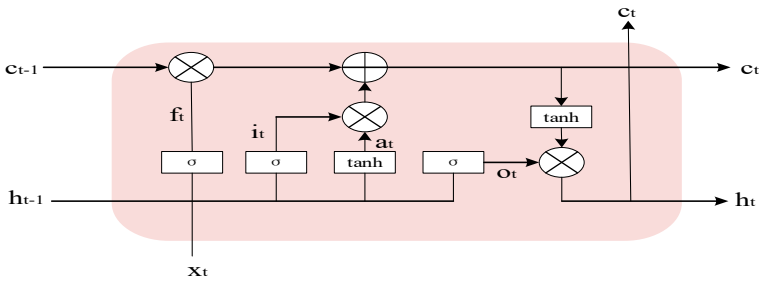


Figure 8. Internal Structure of LSTM Cell.

2.3. TextCNN

The TextCNN model proposed by Kim [5] consists of one-dimensional convolutional layers and one-dimensional max-pooling layers. Its input is a text matrix composed of word vectors, and the width of the convolutional kernels is the same as the width of the text matrix. In other words, the width of the convolutional kernels is equal to the dimension of the word vectors. The convolutional kernels only move vertically. The model structure is illustrated in Figure 9.

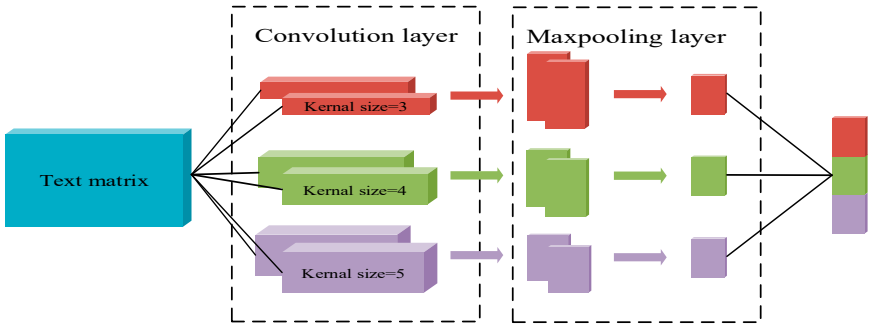


Figure 9. TextCNN Operation.

After convolving a text vector with a kernel, the resulting output is a vector with a shape of (sentence length − kernel height + 1, 1). The calculation of the convolution operation is shown in Equation (18).

$$c = f(w_i \times M + b) \tag{18}$$

In the equation, 'c' represents the feature vector obtained from the convolutional kernel output, 'f' is the activation function of the convolutional layer, 'b' stands for the bias term, 'w_i' denotes convolutional kernels of different sizes, and 'M' represents the text matrix.

Due to the varying sizes assigned to the convolutional kernels 'w_i', they can extract text features at different scales. Therefore, the text features 'k' preserved after passing through the pooling layer represent the multi-scale characteristics of the text. The calculation of the pooling operation is demonstrated in Equation (19).

$$k = \max(c_1, c_2, \ldots, c_n) \tag{19}$$

Subsequently, the obtained features '*k*' from each operation are concatenated to form a text feature vector, which serves as the input for the subsequent stage of operations.

2.4. Self-Attention

The design of the Self-Attention [18] mechanism is inspired by human visual processing, where a preview of the global information is taken to gauge the attention towards local details. This approach allows for a better focus on key information while selectively suppressing irrelevant data. The structure of the mechanism is depicted in Figure 10.

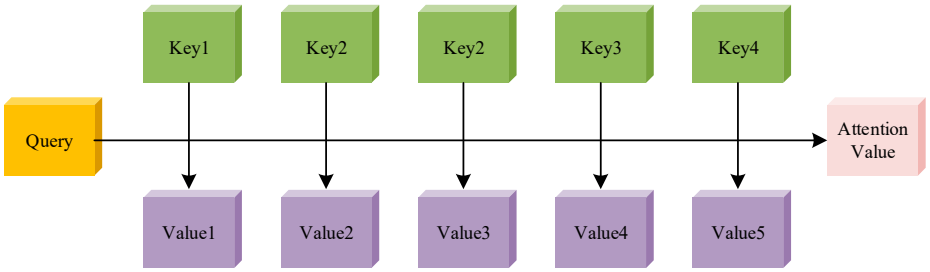


Figure 10. Self-Attention calculation process.

Self-Attention is fundamentally a mapping relationship composed of '*Query*', '*Key*', and '*Value*' components for each local information. Its primary computation process can be summarized in the following steps:

Step 1: Compute the similarity between the '*Query*' of a specific local information and each '*Key*' to obtain corresponding weights, which can be calculated using Equation (20).

$$similar(Q, K) = QK^T \tag{20}$$

Step 2: Normalize the weight values through '$Softmax$' function to prevent occurrences of extremely large or small values. This normalization process can be computed according to Equation (21).

$$a_i = softmax(similar(Q, K)) \tag{21}$$

Step 3: Compute the final '*Attention*' by performing a weighted sum of 'a_i' with the corresponding '*Value*' components, which can be calculated using Equation (22).

$$Self - Attention(Q, K, V) = \sum a_i V \tag{22}$$

3. Methodology

3.1. Noise Injection and Adversarial Training

Given the limited semantic information in short texts coupled with their high dimensionality, overfitting can become a challenge. In the context of deep learning models, various approaches are typically employed to mitigate overfitting and enhance generalization ability:

1. Utilizing dropout: This involves randomly deactivating different weights during training.
2. Leveraging adversarial networks for noise injection: Adversarial networks are used to generate noisy data, introducing necessary perturbations.

These strategies collectively aim to prevent overfitting and enhance the model's ability to generalize.

The principle of dropout involves randomly deactivating a subset of parameters during each training iteration. In each training cycle, a random selection of parameters is chosen to be hidden, with their values set to zero. Since different parameters are hidden in each training iteration, it is akin to multiple instances of different sub-networks being combined and stacked. This results in diverse computational outcomes being compounded and outputted, leading to a richer representation of information features. In comparison to

the original network, the network at this stage is not only simplified but also captures more stochastic factors, consequently reducing the occurrence of overfitting.

The GAN [33,34] is a generative modeling technique based on differentiable generative networks, rooted in the principles of game theory. Miyato [35] introduced adversarial training and virtual training to semi-supervised text classification tasks, effectively mitigating the occurrence of overfitting. Chen [36] combined the attention mechanism with a multitasking-focused GAN network. Under the attention mechanism, a subset of the original text is allocated for adversarial training. Within the GAN architecture, the generator and discriminator compete with each other to achieve a "Nash equilibrium", as illustrated in Figure 11.

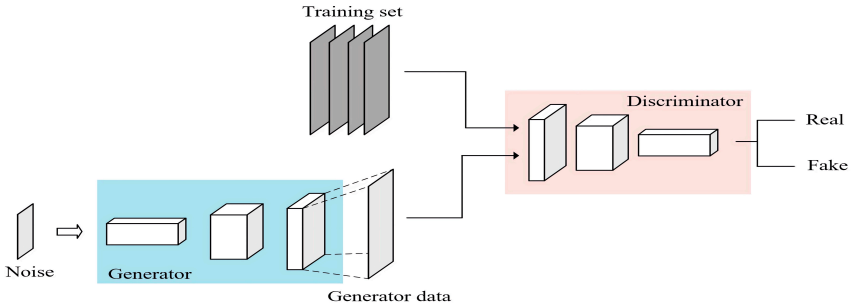


Figure 11. Architecture of GAN Network Model.

The generator captures the underlying distribution of the training data, aiming to generate synthetic data that conforms to the same distribution. It expects the discriminator to classify the generated data as real training data. The discriminator, guided by the standards set by the training data, encourages the generator to produce more "realistic" synthetic data. Through the adversarial interplay and continuous self-improvement between these two networks, the objective is to generate training data that closely resembles real data. The loss function of the adversarial network is defined by Equation (23):

$$\min_G \max_D V(D,G) = E_{X-P_{data(x)}}[\log D(x)] + E_{X-P_{g(x)}}[\log(1-D(G(z))] \quad (23)$$

In Equation (21): 'x' represents the real samples input into the network; 'E' represents the expectation of inputting real samples; '$D(\cdot)$' signifies the output values of the discriminator's judgment; '$G(\cdot)$' signifies the output values of the generator.

To address the limitations of having a single perturbation parameter, fixed normalization, and an inability to accurately control the noise magnitude within the model, a combination of adversarial training network and dropout is employed. This approach generates a substantial number of noisy samples, enhancing both model performance and robustness, thereby mitigating the risk of overfitting.

3.2. Model Architecture

To emphasize the essential information for sentiment analysis within short texts, this paper introduces a novel text sentiment analysis BiLSTM-TCSA model for binary classification of sentences, which integrates word embedding techniques with BiLSTM, TextCNN, and Self-Attention. The proposed model utilizes Glove Vectors for word Representation (GloVe) [37] pre-trained word embeddings derived from a vast corpus to represent text vectors, and further fine-tunes these embeddings during the entire training process in conjunction with the dataset.

Prior to training, a certain quantity of perturbed texts is generated using a GAN and incorporated into the training dataset. This approach not only enhances the robustness of the model but also effectively addresses overfitting issues. In the learning of contextual dependency, a dual-layer BiLSTM mechanism that incorporates the Resnet concept is employed to fuse initially independent word embeddings. This mechanism establishes connections between contextual semantic information, yielding comprehensive global semantic

features. For the learning of local semantic relationships, a TextCNN with convolutional kernels of sizes (3, 4, 5) is employed. The convolutional operations are followed by pooling, resulting in local semantic features that are concatenated. The Self-Attention mechanism is then employed to enhance the sensitivity of sentiment word features while suppressing non-keyword features. This enhances the model's focus on text information and improves classification efficiency.

Regarding the overall model architecture, during the training process, the IPSO is employed to iteratively search for the optimal combination of model hyperparameters. This ensures that the model achieves maximum convergence. The holistic structure of the model is illustrated in Figure 12.

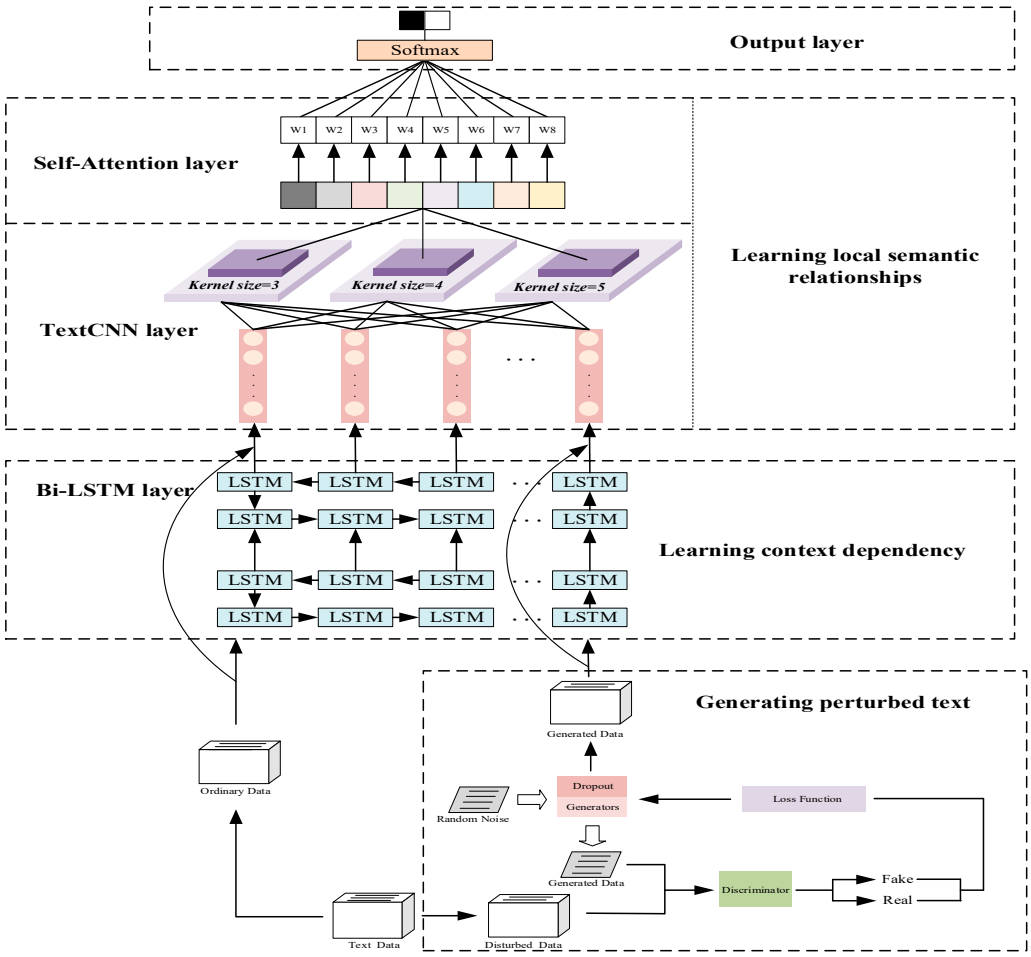


Figure 12. Architecture of BiLSTM-TCSA.

4. Experiments and Analysis

4.1. Experimental Environment

The experiments were conducted using the open-source deep learning framework Keras (Version-2.4.3), implemented using the Python 3.8 programming language on the PyCharm platform. The experimental setup was as follows: the operating system was Windows 10, the CPU was AMD Ryzen 7 5800H, and the GPU was NVIDIA GeForce GTX 1650. A summary of the experimental environment information is presented in Table 3.

Table 3. Experimental environment.

Device	Version
Operating System	Win10 64-bit
CPU	AMD Ryzen 7 5800H
RAM	16G
GPU	GTX1650
Programming Language	Python 3.8
Integrated Development Environment	Pycharm Community Edition
Deep Learning Framework	Kreas 2.4.3

4.2. Datasets

To validate the rationality and effectiveness of the BiLSTM-TCSA model, four widely used public datasets were selected: the IMDB movie review dataset, the MR English movie review dataset, the SST-2 dataset from the Stanford Sentiment Treebank, and the Hotel Reviews dataset containing user reviews from renowned hotels worldwide.

(1) The IMDB: This movie review dataset comprises user reviews from the IMDB website about movies. For the experiment, all 50,000 records from this dataset were obtained.
(2) The MR: This dataset is a binary-class English movie review dataset, containing an equal number of positive and negative sentiment samples, totaling 10,662 instances.
(3) The SST-2: This dataset originates from the Stanford Sentiment Treebank and is designed specifically for binary sentiment classification tasks, comprising a total of 70,044 sentiment-labeled comments.
(4) The Hotel Reviews: This dataset, hereinafter referred to as the HR dataset, consists of user-generated polarity sentiment reviews about renowned hotels worldwide, collected by the Whale Community. After eliminating irrelevant comments, this dataset comprises a total of 877,640 valuable entries.

However, to enhance the robustness of the model, several sets of perturbed texts were extracted from the dataset and input into the GAN to generate perturbed texts. The number of generated perturbed texts accounted for 1/10 of the dataset's sample size. Subsequently, these perturbed texts were combined with the original dataset. Following the principle of maintaining a balanced proportion of positive and negative samples, a training set to testing set ratio of 7:3 was established, and this combined dataset was used for model training. The relevant statistical information for each dataset is presented in Table 4.

Table 4. Information for each dataset.

Datasets	Number of Samples	Average Length	Number of Train Data	Number of Test Data
IMDB	50,000 + 5000	282	38,500	16,500
MR	10,662 + 1066	20	8210	3518
SST-2	70,042 + 7004	13	53,932	23,114
HR	877,640 + 87,764	24	668,083	286,321

4.3. Metrics

In the sentiment analysis experiments, this study employs accuracy, recall, and F1 score as evaluation metrics for the models. The specific calculation formulas are as follows in Equations (24)–(26):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{24}$$

$$Recall = \frac{TP}{TP + FP} \tag{25}$$

$$F1\_Score = \frac{2TP}{2TP + FP + FN} \tag{26}$$

where TP (True Positive) represents the number of samples correctly classified as positive sentiment, FP (False Positive) represents the number of samples incorrectly classified as positive sentiment, FN (False Negative) represents the number of samples incorrectly classified as negative sentiment, and TN (True Negative) represents the number of samples correctly classified as negative sentiment.

4.4. Hyperparameter Optimization Based on IPSO

The parameters for the IPSO were initially set with a particle count of $m = 15$. The dimension of the optimization hyperparameters was defined as $d = 5$, and the experiment was conducted over a total of $k = 100$ iterations.

The optimization of hyperparameters for the network model is elaborated upon below. In this study, the network model is constructed within the Keras framework. It employs a two-layer BiLSTM and utilizes the Cross Entropy Loss function as its loss function. The optimizer of choice is Stochastic Gradient Descent (SGD). The hyperparameters selected for optimization encompass the dropout rate, BiLSTM hidden layer dimension (hd), learning rate, batch size, and convolutional kernel stride.

(1) Dropout rate, denoted as 'd_p', representing the random deactivation rate of the dropout layer in the generators of GAN, with a range of $d_p \in (0,\ 1)$;
(2) BiLSTM hidden layer dimensions, denoted as 'h^d', indicating the dimensionality of the output vectors from the BiLSTM layer, with a range of $h^d \in [50,\ 200]$;
(3) Learning rate, denoted as 'lr', which controls the magnitude of updates to the internal parameters during model training, with $lr \in [0.0001,\ 0.01]$;
(4) Batch size, indicating the number of samples used in each training iteration, with $batch\_size \in \{16,\ 32,\ 64,\ 128\}$;
(5) Convolutional kernel strides, denoted as '*strides*', representing the length of each movement of the convolutional kernel in the TextCNN layer, with $strides \in \{1,\ 2,\ 3,\ 4\}$;

In the process of fine-tuning deep learning models using IPSO, it is necessary to abstract the model as a '*model*' function. The fitness function for this deep learning model typically adopts the loss function used by the model. Here, the fitness function is set to the '*cross_enterpory*'. The operational procedure is illustrated in Equations (27) and (28):

$$cross\_enterpory = \begin{cases} model - 1 & sample\_label = 1 \\ model - 0 & sample\_label = 0 \end{cases} \tag{27}$$

$$model = model(BiLSTM, GAN, TextCNN, Learning\_rate, batch\_size) \tag{28}$$

Upon observation, it is apparent that, although the model includes a self-attention mechanism not explicitly listed in the '*model*' function, the parameters within this mechanism adapt through backpropagation during iteration. In contrast, *BiLSTM*, *GAN*, and *TextCNN* each have their own set of hyperparameters that require optimization. During the iterative process, the designated learning rate and batch size are also chosen as hyperparameters for optimization, as indicated in Equations (29)–(31):

$$BiLSTM = BiLSTM(h^d) \tag{29}$$

$$GAN = GAN(d_p) \tag{30}$$

$$TextCNN = TextCNN(strides) \tag{31}$$

The optimal hyperparameter search results obtained through IPSO are presented in Table 5.

Table 5. Results of Hyperparameter Optimization (IMDB/SST-2/MR/HR).

Hyperparameter	Value
d_p	0.38/0.42/0.37/0.44
h^d	141/97/94/101
lr	0.0079/0.0072/0.0068/0.0078
batch_size	32/32/32/64
strides	1/1/1/1

4.5. Comparative Model Experiments and Results Analysis

To validate the performance of our model and analyze the impact of the GAN mechanism, considering the characteristics of text structure and sentiment analysis tasks, this study conducted comparative experiments on four datasets: IMDB, MR, SST-2, and HR. Our model was compared with classical models such as FastText, TextCNN-MultiChannel, DCNN, RCNN, BiLSTM+EMB_Att, BiLSTM-cnn, MLACNN, and MC-AttCNN-AttBiGRU, as well as our model without the GAN. The details of the comparative models are as follows:

(1) FastText [38]: an open-source model by Facebook which is a rapid text classification approach. Its fundamental idea involves the summation of N-gram vectors of the text, followed by averaging to obtain the vector representation of the text. This vector is then fed into a softmax classifier to yield the classification outcome.
(2) TextCNN-MultiChannel [5]: This model, also known as the TextCNN model proposed by Kim, is a multi-channel convolutional neural network. It utilizes convolutional kernels with window sizes of 3, 4, and 5 in the convolutional layers to extract local text features of various scales. These features are then concatenated after undergoing max-pooling, followed by classification using the Softmax operation.
(3) DCNN [39]: The DCNN employs wide convolutions for feature extraction, allowing it to effectively utilize the edge information present in the text.
(4) RCNN [40]: The RCNN employs a recurrent structure to learn word representations while simultaneously capturing contextual information as much as possible. Additionally, it incorporates maximum pooling to automatically identify the pivotal role of words in text classification, thereby capturing critical semantics within the text.
(5) BiLSTM+EMB_Att [41]: The BiLSTM+EMB_Att model employs an attention mechanism to directly learn the weight distribution of each word's sentiment tendency from the foundational word vectors. It then utilizes a BiLSTM to capture the semantic information within the text, followed by a fusion process for classification.
(6) BiLSTM-CNN [17]: The BiLSTM-CNN model initially employs BiLSTM to extract contextual semantic information. Subsequently, it utilizes multiple convolutional layers to extract local semantics and then fuses them. The classification is ultimately performed using the Softmax function.
(7) MLACNN [42]: The MLACNN model employs CNN to extract local semantic information and utilizes LSTM-Attention to obtain global semantic information. The outputs from both components are fused to achieve classification.
(8) MC-AttCNN-AttBiGRU [21]: This model incorporates the Attention mechanism over word embeddings. It employs both TextCNN and BiGRU to capture local and global features, respectively. The extracted features from both components are then fused and used for classification.

Analysis of the comparative experimental results in Table 6 reveals that, on a macroscopic level, individual deep learning models perform consistently lower across various metrics compared to deep stacked models. This discrepancy can be attributed to the heightened emphasis of deep-layered network models in fully harnessing features, showcasing a more pronounced utilization compared to singular models. At a micro level, the TextCNN-MultiChannel model outperforms FastText across various datasets in terms of accuracy. This is attributed to the fact that the FastText model, relying on text N-gram vectors and simple averaging of word vectors, not only fails to conduct feature extraction but also

introduces substantial noise. In contrast, the TextCNN-MultiChannel model employs three different sizes of convolutional kernels for localized feature extraction, resulting in superior and noise-free features. Building upon the foundation of TextCNN-MultiChannel, the DCNN model augments the convolutional kernel width to incorporate edge information within the text, achieving the utilization and comprehension of global textual information. As a result, DCNN and TextCNN-MultiChannel exhibit comparable performances in practical experiments. RCNN focuses not only on temporal continuity but also effectively employs CNN for feature extraction, merging global and local features. This compensates for DCNN's limitations in temporal feature extraction capability, resulting in slightly improved performance compared to DCNN. Both BiLSTM+EMB_Att and BiLSTM-CNN belong to stacked networks, and they exhibit significant advantages in feature extraction and utilization compared to single-network models. BiLSTM+EMB_Att combines BiLSTM with the attention mechanism to effectively extract keywords relevant to sentiment analysis. BiLSTM-CNN, on the other hand, first extracts contextual semantic features and then employs CNN for local convolutional operations on global features. While their approaches differ, both models achieve similar outcomes, with accuracies of 84.49% and 85.75%, 88.84% and 88.57%, 81.32% and 80.60%, and 89.03% and 89.69% on the four datasets, respectively. MLACNN synthesizes the strengths of both BiLSTM+EMB_Att and BiLSTM-CNN. It incorporates TextCNN for local semantic understanding and BiLSTM for global context comprehension, further utilizing attention to appropriately allocate weights to features. In comparison to BiLSTM-CNN, MLACNN showcases a roughly 1% improvement across three metrics. Differentiating itself from MLACNN, MC-AttCNN-AttBiGRU integrates a dual-channel processing mechanism. After connecting word vectors with attention, it leverages TextCNN and BiGRU for parallel processing. Experimental results indicate that MC-AttCNN-AttBiGRU, with its dual-channel processing approach, outperforms MLACNN in terms of accuracy.

Table 6. Results of Comparative Experiments for four datasets (Accuracy/Recall/F1).

Models	IMDB	SST-2	MR	HR
FastText	81.25/80.95/82.71	85.03/84.82/84.91	76.81/77.03/76.75	84.32/84.26/85.71
TextCNN-MultiChannel	82.33/81.27/80.56	87.12/86.91/87.24	77.94/76.02/78.13	84.79/83.09/83.26
DCNN	81.99/83.26/82.03	86.83/86.03/86.80	77.52/76.59/77.46	86.76/84.23/83.79
RCNN	82.41/81.71/80.26	87.82/88.41/87.74	80.28/81.99/80.25	87.91/85.99/85.41
BiLSTM+EMB_Att	84.49/83.02/85.13	88.84/87.56/88.65	81.32/82.31/81.37	89.03/87.23/88.41
BiLSTM-CNN	85.75/84.26/84.51	88.57/89.21/88.41	80.60/79.21/80.47	89.69/87.29/87.03
MLACNN	86.47/86.74/85.02	89.83/89.95/89.91	82.51/81.03/81.89	90.24/89.46/88.73
MC-AttCNN-AttBiGRU	88.39/87.09/87.38	90.35/90.04/90.32	83.38/82.71/83.30	92.35/91.23/91.45
BiLSTM-TCSA(Ours)	**91.38/92.15/91.52**	**91.74/91.04/92.20**	**85.49/84.76/84.21**	**94.59/92.71/92.17**

In comparison to the aforementioned models, our proposed model not only incorporates a module for processing both global and local features through Self-Attention but also enhances its robustness by introducing a perturbed text generation mechanism based on GAN and integrating the generated perturbed text into the training process. This augmentation further enhances the model's fitting capacity and resilience against disturbances. Moreover, by utilizing an IPSO during the training process, the model's relevant hyperparameters are tuned to their optimal values, fully unleashing the model's performance potential. In experimental validation, our model achieves accuracy rates of 91.23%, 91.74%, 85.49%, and 94.59% on the four datasets, respectively. Additionally, the Recall and F1 scores of our model outperform those of the other comparative models.

4.6. Impact Analysis of GAN

To assess the impact of GANs on the model's robustness against perturbations, this section employs several comparative models from the previous section as well as the

model presented in this paper. The experimental results are visualized in Table 7 to facilitate analysis.

Table 7. The optimized hyperparameters for each model (without GAN/with GAN).

Models	IMDB	SST-2	MR	HR
FastText	81.25/82.03	85.03/85.99	76.81/77.79	84.32/85.72
TextCNN-MultiChannel	82.33/83.16	87.12/87.81	77.94/79.03	84.79/86.23
DCNN	81.99/83.26	86.83/87.71	77.52/78.49	86.76/88.01
RCNN	82.41/83.44	87.82/88.52	80.28/81.31	87.91/89.02
BiLSTM-TCSA(Ours)	90.24/**91.38**	90.88/**91.74**	84.36/**85.49**	93.24/**94.59**

In analyzing the impact of GAN mechanisms on model performance, we conducted comparative experiments. Through these comparative experiments, it was observed that perturbed text generated during model training with GAN mechanisms can effectively identify ambiguous text in sentiment analysis, resulting in an approximate 1% improvement in accuracy across all models on the dataset. This partially mitigates the interference caused by perturbed text. Additionally, the GAN mechanism in our proposed model also performed impressively, leading to accuracy improvements of 1.14%, 0.86%, 1.13%, and 1.35% on the four datasets, respectively.

4.7. Ablation and Validation Experiments

To validate the effectiveness of the IPSO, we selected BiLSTM, LSTM, TextCNN, and Attention as control groups, combined with the comprehensive algorithm. A comparative experiment was conducted against single models without the IPSO. The optimized hyperparameters chosen for each model are listed in Table 8.

Table 8. The optimized hyperparameters for each model.

Model	Hidden Layer Dimension	Learning Rate	Batch_Size	Strides	Dropout
LSTM	☑	☑	☑		
BiLSTM	☑	☑	☑		
LSTM-TextCNN	☑	☑	☑	☑	
BiLSTM-TextCNN	☑	☑	☑	☑	
LSTM-Attention	☑	☑	☑		
BiLSTM-Attention	☑	☑	☑		
TextCNN-Attention		☑	☑	☑	
BiLSTM-TCSA (Ours)	☑	☑	☑	☑	☑

After selecting the hyperparameters that need optimization for the models, we conducted ablation experiments to demonstrate the rationale and effectiveness of the model incorporating the IPSO in hyperparameter selection. The accuracy of the experimental results is shown in Figures 13–16.

From the above bar chart, it is evident that the IPSO has effectively identified suitable hyperparameters for each respective model, fully unleashing their performance potential. The accuracy of each model across the four datasets has been enhanced by an average of approximately 0.5% to 1%, with LSTM-TextCNN, BiLSTM-TextCNN, LSTM-Attention, and the proposed model experiencing even greater average improvements of 1.18%, 1.05%, 1.13%, and 1.065%, respectively. For our model, BiLSTM-TCSA, comparative experiments were also conducted. The performance across four datasets reveals that utilizing the IPSO algorithm for hyperparameter tuning outperforms manual tuning in terms of accuracy by 0.89%, 1.58%, 0.61%, and 1.18%, respectively. This is because manual tuning relies on brute-force search, which becomes challenging when dealing with continuous variables, leading to an infinite number of hyperparameter combinations. In such cases, the

quality of manually selected hyperparameters largely depends on luck. In contrast, the IPSO algorithm iteratively updates and shares particle information within the swarm, ultimately finding the optimal combination in the search space. This highlights how the IPSO algorithm outperforms manual empirical selection when it comes to choosing model hyperparameters.

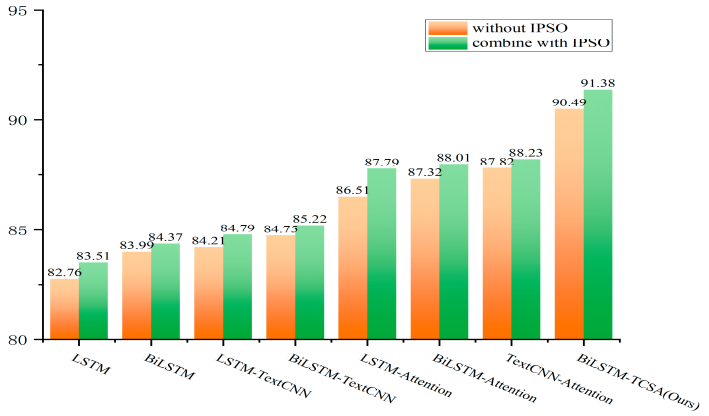


Figure 13. Analysis of IPSO's Impact on Model Performance (IMDB).

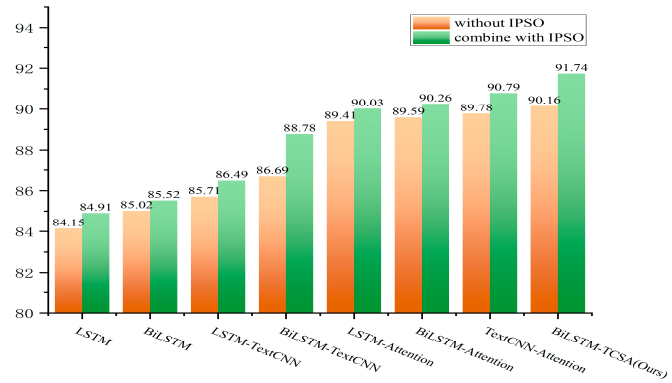


Figure 14. Analysis of IPSO's Impact on Model Performance (SST-2).

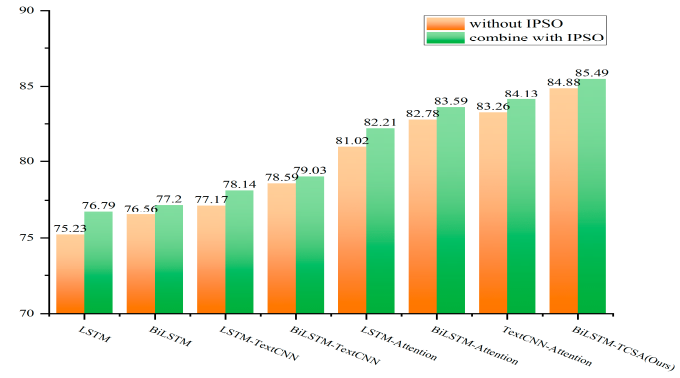


Figure 15. Analysis of IPSO's Impact on Model Performance (MR).

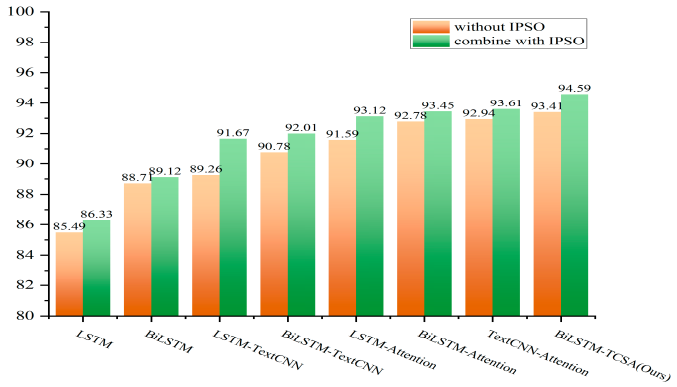


Figure 16. Analysis of IPSO's Impact on Model Performance (HR).

5. Conclusions and Future Work

The proposed deep learning model for short-text sentiment analysis, based on the IPSO, effectively leverages the IPSO approach to extract deep textual features. Moreover, the model's robustness against disturbances is enhanced through the generation of a large amount of perturbed text using a GAN model. Experimental results indicate that utilizing the IPSO yields varied effects based on different combinations of model hyperparameters. The algorithm dynamically and adaptively adjusts inertia weights and learning factors, which not only prevents particles from becoming trapped in local optima to some extent, but also rapidly and effectively facilitates complete model convergence. This feature enables the full unleashing of model performance potential.

In the subsequent stages of experimental research, further efforts will be dedicated to refining the PSO. Additionally, consideration will be given to conducting horizontal comparisons with other parameter optimization algorithms, such as Bayesian optimization, genetic algorithms, and simulated annealing. The aim is to achieve further enhancements in metrics such as the model's prediction accuracy.

Author Contributions: Y.P. and Y.Y. contributed to the conception of this study; Y.Y. performed the experiment, contributed significantly to analysis and wrote the manuscript; D.W. helped perform the analysis with constructive discussions. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the National Natural Science Foundation of China (No. 61966017) and by the Natural Science Foundation project of JiangXi province (No. 20224BAB202013 and No. 20212BAB202017).

Data Availability Statement: This study is an experimental analysis of a publicly available dataset.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Vries, A.; Mamoulis, N.; Nes, N. Efficient k-NN search on vertically decomposed data. In Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, WI, USA, 3–6 June 2002; pp. 322–333.
2. Chen, Z.; Shi, G.; Wang, X. Text Classification Based on Naive Bayes Algorithm with Feature Selection. *Int. J. Inf.* **2012**, *15*, 4255–4260.
3. Wang, J.H.; Yu, H.; Chan, W. Automatic text Classification based on KNN+ Hierarchical SVM. *Comput. Appl. Softw.* **2016**, *33*, 38–41.
4. Rojas, B.; Maria, L. Deep learning for sentiment analysis. *Ling. Linguist. Compass* **2016**, *10*, 205–212.
5. Kim, Y. Convolutional Neural Networks for Sentence Classification. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, Doha, Qatar, 25–29 October 2014; Association for Computational Linguistics: Stroudsburg, PA, USA, 2014; pp. 1746–1751.
6. Conneau, A.; Schwenk, H.; Barrault, L. Very deep convolutional networks for text classification. *arXiv* **2016**, arXiv:1606.01781v1.
7. Le, H.T.; Cerisara, C.; Denis, A. Do convolutional networks need to be deep for text classification? *arXiv* **2017**, arXiv:1707.04108.
8. Johnson, R.; Zhang, T. Deep pyramid convolutional neural networks for text categorization. In Proceedings of the 55th Annual Meetings of the Association for Computational Linguistics, Vancouver, BC, Canada, 30 July–4 August 2017; pp. 562–570.

9. Guo, J.; Yue, B.; Xu, G. An enhanced convolutional neural network model for answer selection. In Proceedings of the 26th International Conference on World Wide Web Companion, Perth, Australia, 3–7 April 2017; pp. 789–790.
10. Wang, H.; He, J.; Zhang, X. A short text classification method based on n-gram and CNN. *Chin. J. Electron.* **2020**, *29*, 248–254. [CrossRef]
11. Irsoy, O.; Cardie, C. Opinion Mining with Deep Recurrent Neural Networks. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 720–728.
12. Xin, W.; Liu, Y.; Sun, C. Predicting Polarities of Tweets by Composing Word Embeddings with Long Short Term Memory. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, Beijing, China, 26–31 July 2015; pp. 1343–1353.
13. Hochreiter, S.; Schmidhuber, J. Long short-term memery. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
14. Gers, F.A.; Schermidhber, J.; Cummins, F. Learning to forget: Continual prediction with LSTM. *Neual Comput.* **2000**, *12*, 2451–2471. [CrossRef]
15. Cho, K.; Van Merrienboer, B.; Gulcehre, C. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv* **2014**, arXiv:1406.1078.
16. Tai, K.S.; Socher, R.; Manning, C.D. Improved semantic representations from tree-structured long short-term memory networks. *arXiv* **2015**, arXiv:1503.00075.
17. Zhao, H.; Wang, L.; Wang, W.J. Text sentiment analysis based on serial hybrid model of bi-directional long short-term memory and convolutional neural network. *J. Comput. Appl.* **2020**, *40*, 16–22.
18. Vaswani, A.; Shazeer, N.; Parmar, N. Attention is all you need. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 5998–6008.
19. Yang, Z.; Yang, D.; Dyer, C. Hierarchical attention networks for document classification. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego, CA, USA, 12–17 June 2016; pp. 1480–1489.
20. Zhou, Y.; Xu, J.; Cao, J. Hybrid attention networks for Chinese short text classification. *Comput. Syst.* **2017**, *21*, 759–769. [CrossRef]
21. Cheng, Y.; Yao, L.B.; Zhang, G.H.; Tang, T.W.; Xiang, G.X.; Chen, H.M.; Feng, Y.; Cai, Z. Text Sentiment Orientation Analysis of Multi-Channels CNN and BiGRU Based on Attention Mechanism. *J. Comput. Res. Dev.* **2020**, *57*, 2583–2595.
22. Matthias, F.; Katharina, E.; Stefan, F. Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. *arXiv* **2021**, arXiv:2007.04074v2.
23. Wang, Y.; Yan, Y.; Li, Z. Efficient and Robust Auto-tuning of Deep Learning Hyperparameters with Gaussian Processses. *JMLR* **2023**, *24*, 1–99.
24. Jeremy, B.; Chirs, M. Automatic Gradient Descent: Deep Learning without Hyperparameters. *arXiv* **2023**, arXiv:2304.05187.
25. Yapici, H.; Cetinkaya, N. A new meta-heuristic optimizer: Pathfinder algorithm. *Appl. Soft Comput.* **2019**, *78*, 545–568. [CrossRef]
26. Mirjalili, S.; Hatamlou, A. Multi-verse optimizer: A nature-inspired algorithm for global optimization. *Neural Comput. Appl.* **2016**, *27*, 459–513. [CrossRef]
27. Mirijalili, S. Moth-flame optimization algorith: A novel nature-inspired heuristic paradigm. *Knowl.-Based Syst.* **2015**, *89*, 228–249. [CrossRef]
28. Abualigah, L.; Diabat, A.; Mirjalili, S. The arithmetic optimization algorithm. *Comput. Methods Appl. Mech. Eng.* **2021**, *376*, 133609. [CrossRef]
29. Kennedy, J.; Eberhart, R. Particle Swarm Optimization. In Proceedings of the Icnn95-International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; pp. 1942–1948.
30. Wang, J.J.; Tufan, K. Parameter Optimization of Interval Type-2 Fuzzy Neural Network Based on PSO and BBBC Methods. *IEEE/CAA J. Autom. Sin.* **2019**, *1*, 247–257. [CrossRef]
31. Shafipour, M.; Rashno, A.; Fadaei, S. Particle distance rank feature selection by particle swarm optimization. *Expert Syst. Appl.* **2021**, *185*, 115620. [CrossRef]
32. Graves, A.; Schmidhuber, J. Framewise phoneme classification with bidirectional LSTM networks. In Proceedings of the IEEE International Joint Conference on Neural Networks, Montreal, QC, Canda, 31 July–4 August 2005; pp. 2047–2052.
33. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M. Generative Adversarial Nets. In Proceedings of the International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2014; pp. 2672–2680.
34. Liang, J.J.; Wei, J.J.; Jiang, Z.F. Generative Adversarial Networks GAN Overview. *J. Front. Comput. Sci. Technol.* **2020**, *14*, 1–17.
35. Miyato, T.; Dai, A.M.; Goodfellow, I. Adversarial training methods for semi-supervised text classification. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017; pp. 1–11.
36. Chen, R.L. Attention-based adversarial multi-task review text classification. Master's Thesis, Dalian University of Technology, Dalian, China, 10 May 2019.
37. Pennington, J.; Socher, R.; Manning, C. Glove: Global Vectors for Word Representation. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, Doha, Qatar, 25–29 October 2014; pp. 1532–1543.
38. Joulin, A.; Grave, E.; Bojanowski, P. Bag of tricks for efficient text classification. *arXiv* **2016**, arXiv:1607.07159v3.
39. Nal, K.; Edward, G.; Phil, B. A convolutional neural network for modelling sentences. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, Baltimore, MD, USA, 23–25 June 2014; pp. 655–665.
40. Lai, S.; Xu, L.; Liu, K. Recurrent convolutional neural networks for text classification. In Proceedings of the 29th Conference on Artififcal Inteligence, Austin, TX, USA, 25–30 January 2015; pp. 2267–2273.

41. Guan, P.F.; Li, B.A.; Lv, X.Q.; Zhou, J.S. Attention Enhanced Bi-directional LSTM for Sentiment Analysis. *JCIP* **2019**, *33*, 105–111.
42. Teng, J.B.; Kong, W.W.; Tian, Q.X.; Wang, Z.Q. Text Classification Method Based on LSTM-Attention and CNN Hybrid Model. *Comput. Eng. Appl.* **2021**, *57*, 126–133.