Hindawi
Scientific Programming
Volume 2022, Article ID 6519909, 13 pages
https://doi.org/10.1155/2022/6519909

*Research Article*

# A Hyperparameter Optimization Algorithm for the LSTM Temperature Prediction Model in Data Center

**Simin Wang**,[1,2] **Chunmiao Ma**,[1] **Yixuan Xu**,[1] **Jinyu Wang**,[1] **and Weiguo Wu**[1]

[1]*School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China*
[2]*Xi'an Polytechnic University, Xi'an 710048, China*

Correspondence should be addressed to Weiguo Wu; wgwu@xjtu.edu.cn

Received 22 June 2022; Revised 1 September 2022; Accepted 26 November 2022; Published 12 December 2022

Academic Editor: Jianping Gou



As the main tool to realize data mining and efficient knowledge acquisition in the era of big data, machine learning is widely used in data center energy-saving research. The temperature prediction model based on machine learning predicts the state of the data center according to the upcoming tasks. It can adjust the refrigeration equipment in advance to avoid temperature regulation lag and set the air conditioning temperature according to the actual demand to avoid excessive refrigeration. Task scheduling and migration algorithm based on temperature prediction can effectively avoid hot spots. However, the choice of hyperparameter of machine learning model has a great impact on its performance. In this study, a hyperparameter optimization algorithm based on MLP is proposed. On the basis of trying certain hyperparameters, the MLP model is used to predict the value of all hyperparameters' space, and then, a certain number of high-quality hyperparameters are selected to train the model repeatedly. In each iteration, the amount of training data decreases gradually, while the accuracy of the model improves rapidly, and finally, the appropriate hyperparameter are obtained. We use the idea of mutation in the genetic algorithm to improve the probability of high-quality solutions and the loss function weighting method to select the solution with the best stability. Experiments are carried out on two representative machine learning models, LSTM and Random Forest, and compared with the standard Gaussian Bayes and Random Search method. The results show that the method proposed in this study can obtain high-precision and high-stability hyperparameter through one run and can greatly improve the operation efficiency. This algorithm is not only effective for LSTM but also suitable for other machine learning models.

## 1. Introduction

The era of big data and cloud computing promotes the rapid expansion of data centers, and the excessive expansion has caused the waste of resources. The energy consumption cost accounts for about 50% of the long-term operation costs of a data center [1]. Among all the energy costs, the energy consumption of refrigeration equipment accounts for about 40%. During the working state, the temperatures of 10% cabinets in a data center are higher than the allowable range of equipment, which is very dangerous and cause the downtime risk. In order to ensure safety, many data centers are set to a lower temperature for a long time, resulting in a large waste of refrigeration energy consumption. Data center server temperature is mainly affected by load and changes dynamically. Therefore, strengthening the temperature monitoring of the data center and timely adjusting the refrigeration equipment can effectively reduce the refrigeration energy consumption [2, 3].

The response cycles of servers and air-conditioning equipment are different. The CPU frequency of modern servers is as high as several GHz, so the response time of servers is very short, which can be as fast as milliseconds. The form of heat exchange in a data center is mainly convection, so the heat flow in the data center is relatively slow. The temperature change of refrigeration equipment may take some time to be sensed by the server. Generally, the period of temperature regulation in the machine room is relatively long, which may be several minutes. The traditional temperature perception method is to dynamically monitor the

air inlet temperature of the server. Once the air inlet temperature of the server is found to exceed the limit, the server load is reduced by decreasing the room temperature or task dynamic migration. However, this feedback based method has the problem of feedback lag. Another popular temperature sensing model is based on Computational Fluid Dynamics (CFD) simulation. Singh et al. [4] used CFD to establish the temperature distribution model of the data center, Chen et al. [5] used CFD to establish the temperature prediction model of the data center and usedsensor data for calibration to achieve good prediction effect. However, CFD simulation has the disadvantages such as large amount of calculation, long running time, and strong dependence on the environment. Experts are required to accurately model the data center computer room. Therefore, it is not suitable for online real-time temperature prediction.

Machine learning algorithm using computers and mathematics methods to learn from data and predict future results, which effectively solve the above problems [6, 7]. Moore et al. [8] used the artificial neural network to establish the thermal model of data center, Li et al. [9] combined physics with machine learning, which used continuous temperature and real-time airflow data to establish prediction models. Xu et al. [10] used the LSTM time series model to establish the data temperature prediction model and adjusted the state of cooling equipment in real time according to the model.

Since the server load in the data center is regular and the temperature is sequential, LSTM can make good use of long-term correlation and short-term correlation to capture long-term memory problems. Therefore, LSTM is very suitable for data center modeling. The temperature prediction model established by machine learning not only has the advantages of fast prediction speed but also does not depend on a specific machine room. However, there is often a problem that the prediction accuracy needs to be optimized.

The performance and computational complexity of machine learning models heavily depend on the hyperparameters, which need to be adjusted to improve the performance of machine learning algorithms in practical applications [11, 12]. Machine learning is a "black box model," and for a given machine learning model, its internal performance (such as gradient information, etc.) cannot be obtained. A lot of machine learning algorithms, such as random forest, LSTM, and support Vector machine, contain at least a dozen or as many as dozens of parameters, and tuning hyperparameter is an onerous task. The traditional manual of tuning hyperparameters can no longer meet the needs.

This paper proposes a hyperparameter optimization method, which uses the MLP model and optimal value perturbation technology to improve the global optimization ability of the MLP. At the end of the training, the stability model of the training process is used to ensure that the selected hyperparameter has excellent performance and good stability.

The machine learning model can ensure that the data center server runs at a safe temperature and effectively guides the adjustment of air conditioning equipment and server task scheduling. Therefore, the performance and prediction accuracy of the model are very important. Choosing better hyperparameters for machine learning model has become the key to accurate energy management in data center, and it is also an important topic in the field of machine learning [13, 14].

## 2. Background and Related Work

The background and related work are described as follows.

*2.1. Long and Short Term Memory Network.* The data center temperature is affected mainly by the server load and the surrounding environment. The changing of loads is the most important factor of server temperature fluctuation. In the closed computer room environment, the heat flow is relatively slow. The heat generation and heat dissipation is a gradual process. Therefore, the time series prediction model is more suitable for data center modeling.

Recurrent neural network (RNN) had a better effect in processing time series data. It uses the internal hidden layer node state to remember any length of input. The result of the output layer is not only related to the current input but also related to the result of the last hidden layer, which means that it has some memory functions. The prediction model of long short term memory network (LSTM) is a variant of RNN. It uses Input Gate, Output Gate, and Forget Gate to adjust whether the previous network state acts on the current calculation, which is conducive to dealing with long distant nodes in the time series. RNN has the problem of gradient explosion or gradient disappearance when dealing with long-distance nodes in time series. Compared with RNN, LSTM has great advantages in processing long-distance nodes of time series. Compared with BP neural network, LSTM has advantages in processing data with high correlation. Under the same input, the output of BP neural network is fixed and has little correlation with input sequence, which makes BP network insensitive to temperature trend. LSTM is a good choice for data center temperature modeling. Figure 1 is the structural diagram of LSTM [15–17].

In Figure 1, $Xn$ and $Yn$ represent input and output data, U, V, and W represent weight coefficients, $hn$ represent hidden layer status, and $hn$ are related to the current input $Xn$ input and the previous R hidden layers. The LSTM adjusts whether the previous network memory state acts on the calculation of the current network through three gates acting on the node. The functions of the three gates are as follows: the Forget Gate controls how much of the previous hidden layer state is retained to the current time; the Input Gate controls how much input $Xn$ of the network at the current time is saved to the hidden layer state $hn$; and the Output Gate controls how many hidden layer states are output to the current time output value $Yn$. The current time step hidden layer states can be expressed as

$$h_n = UX_n + W_{n-1}h_{n-1} + W_{n-2}h_{n-2} + \ldots + W_{n-R}h_{n-R}. \quad (1)$$

In this way, the association with the first $R$ input is realized.

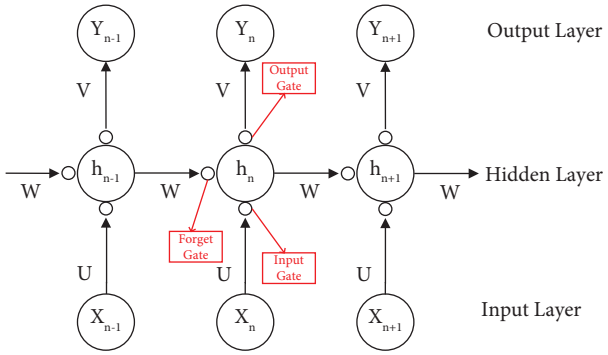


Figure 1: Structure diagram of LSTM.

The three gates constitute the basic unit of LSTM called cell. The basic structure of LSTM neural network cell is shown in Figure 2.

The calculation of Forget Gate $f_n$, Input Gate $i_n$, and Output Gate $o_n$ are shown in formulas (1)–(3):

$$f_n = \delta\left(W_{fx}X_n + W_{fy}Y_{n-1} + b_f\right), \tag{2}$$

$$i_n = \delta\left(W_{ix}X_n + W_{iy}Y_{n-1} + b_i\right), \tag{3}$$

$$o_n = \delta\left(W_{ox}X_n + W_{oy}Y_{n-1} + b_o\right). \tag{4}$$

The calculation of current cell state $h_n$ and output $y_n$ is shown in (5) and (6):

$$h_n = h_{n-1}f_n + i_n\tanh\left(W_cX_n + U_cX_{n-1} + b_c\right), \tag{5}$$

$$y_n = o_n\tanh\left(h_n\right). \tag{6}$$

In formula (2), $\delta$ is the sigmoid activation function acting on the valve. The output value is [0, 1], indicating how much information will be retained in this part. $W$ represents weight, $W_{fx}$ is the weight of the last temporal output information corresponding to the forgetting gate, and $b$ represents the offset amount of this gate. The tanh activation function acts on the cell state and the output cell [18].

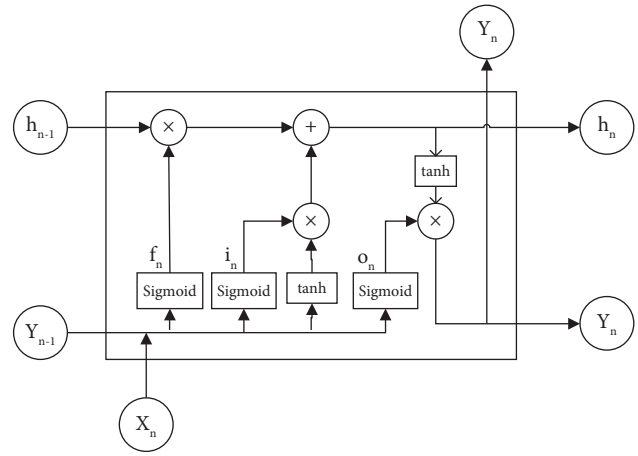


Figure 2: Basic structure of the LSTM neural network cell.

*2.2. Date Center Temperature Prediction Model.* In [10], we used LSTM to establish a data center temperature prediction model. The model input data includes the air inlet and outlet temperature, CPU utilization and fan speed of the server, as well as the air inlet and outlet temperature of the neighboring upper and lower servers of the same cabinet (currently numbered $n$ and neighboring $n-1$ and $n+1$), and CRAC air outlet and return air outlet temperature. According to the corresponding timestamp, each sample is represented by $P(t)$:

$$P(t) = \left(T_{n,\text{in}}, T_{n,\text{out}}, u_{\text{cpu}}, \text{fans}, T_{n+1,\text{in}}, T_{n+1,\text{out}}, T_{n-1,\text{in}}, T_{n-1,\text{out}}, T_{\text{CRAC\_in}}, T_{\text{CRAC\_out}}\right), \tag{7}$$

where $T_{n,\text{in}}$ and $T_{n,\text{in}}$ indicates the temperature at the air inlet and outlet of the server and $u_{\text{cpu}}$ and fans represents the current server CPU utilization and fan speed, $T_{n+1,\text{in}}, T_{n+1,\text{out}}, T_{n-1,\text{in}}, T_{n-1,\text{out}}$ represent the inlet and outlet temperatures of the upper and lower servers neighboring to the server, $T_{\text{CRAC\_in}}, T_{\text{CRAC\_out}}$ indicate the temperature of the return air outlet and air outlet of CRAC, respectively.

The neighboring $R$ samples in the time series are used to predict the server inlet temperature after a period of time interval (this time interval is called the prediction field of view, expressed in $K$), that is, time_Steps = $R$, and the predicted visual field $K$ is an integral multiple of the time interval $s$ for collecting data. If $N$ represents the total number of samples and $C$ represents the dimension of vector $P(t)$, then the data of the input layer $[N, R, C]$ and the input data of a sample are expressed as $< P(t-R+1), P(t-R+2), \cdots, P(t-1), P(t) >$, and the output data pair can be expressed as $< T_{t+k,\text{in}} >$, so we can get the temperature of the server air inlet after $K$ time.

Using this method, we can freely set the prediction duration, select number of groups of historical data to predict the temperature, and establish a flexible data center temperature prediction model.

*2.3. LSTM Model Hyperparameters.* When implementing the LSTM network, the following parameters that have an important impact on the network structure need to be set.

Activation is the activation function. It is set to "tanh" by default and can also be set to "linear," "sigmoid" and "reLU."

Dropout represents that neurons are discarded according to a certain probability to prevent over fitting.

Batch_Size indicates the number of samples for a training. In general, if the Batch_Size is large, the training speed is fast, but the generalization ability of the model is poor.

Epochs represents the number of iterations, that is, the number of complete training using all samples.

Optimizer is the optimizer. When establishing the model, select the appropriate optimizer to calculate the update step.

Timestep is the prediction step. The current output is related to the previous time steps.

Unit represents the number of hidden layers. If the number of hidden layers is set to be too small, the network fitting effect is poor; if it is set as too much, the training time will be prolonged and it is easy to fall into a local minimum. Generally, this parameter is usually set by empirical value.

The setting of hyperparameters has a great impact on the complexity and accuracy of the model. Neural network belongs to the black box model, and the influence of hyperparameters on accuracy can only be obtained after the model is executed. Blind attempt is inefficient, and the parameters settings of Timestep and Unit are also affected by specific data sets. Therefore, an efficient hyperparameters optimization algorithm can better improve the model accuracy and network optimization efficiency.

*2.4. Hyperparameter Optimization Problem.* Hyperparameter optimization (HPO) is the process of choosing a set of hyperparameters that achieve the best performance on the data in a reasonable budget, which is defined as follows.

Let $A$ denote a machine learning algorithm with a configuration space of the overall hyperparameters $Y$. $A$ has $n$ hyperparameters, and the hyperparameters space are $Y = Y_1 \times Y_2 \times Y_3 \ldots \times Y_n$. $A\lambda$ is denoted as $A$ with its hyperparameters $\lambda$, where $\lambda \in Y$; for the given data set $D_{\text{train}}$ and $D_{\text{valid}}$, the optimization problem goal is to find a set of optimal hyperparameters $\lambda$ to minimum the value $L$ [19]:

$$\lambda \in \operatorname{argmin}_{\lambda \in Y} L\left(A_\lambda, D_{\text{train}}, D_{\text{valid}}\right), \tag{8}$$

where $L(A_\lambda, D_{\text{train}}, D_{\text{valid}})$ denotes the loss of the model generated by algorithm $A$ and its parameter $\lambda$ on the training set and the test set.

Hyperparameter optimization is actually a function optimization problem. The previous numerical optimization methods are generally applicable to the problem that the mathematical form of the objective function can be derived. The main ones are the simplex algorithm, gradient descent algorithm, Newton algorithm, quasi-Newton algorithm etc. However, these algorithms are more restricted. Generally, factors such as the setting of the initial value, the type of the objective function, and the constraints of the function will have a greater impact on the results of these algorithms. When there are multiple decision variables in the problem, the benefits and consumption of these algorithms are hardly proportional [20].

In recent years, the widely used hyperparameters' optimization methods mainly include Grid Search, Random Search, Bayes optimization algorithm, and some improved algorithms based on Bayes, such as TPE (Tree-Structured on Parzen Estimator) and SMAC algorithm [21].

Grid Search is an exhaustive method with a certain step size. It divides the parameters into grids within its range, then traverses all possible values of the parameters in the grid, and calculates the corresponding results. In order to find the optimal parameter configuration, it is necessary to continuously reduce the step size of grid division to expand the search space. This will lead to the "exponential explosion" problem, so it is more suitable for small data sets.

Random search is random sampling in the hyperparameters space. When the sample set are enough, the optimal value or approximate optimal value of hyperparameter can be obtained through multiple searches. The Random Search algorithm has been proved by Bergstra in [22], which is superior to the Grid Search algorithm. Its drawback is that the results of each search are quite different, which cannot be considered scientifically. In fact, whether Grid Search or Random Search, it is easy to have very poor results due to the explosion of the number of hyperparameter combinations. This is not an effective hyperparameter optimization method.

Bayes optimization [23, 24] finds the minimum value of the objective function by establishing a proxy function based on the past evaluation results of the objective function. Unlike Random Search or Grid Search, Bayes saves a lot of useless works by referring to previous evaluation results when trying the next hyperparameter. The more typical one is the Bayes optimization algorithm based on the Gaussian process, which uses the Gaussian regression model as the proxy model. In the training process, it uses data to continuously update the proxy model to generate new sampling points and iterates these processes until better expressions are generated. The selection of initial sampling points is very important in this algorithm, and the matrix operation is time-consuming in the process of algorithm evaluation [25].

SMAC (Sequential Model-based Algorithm Configuration) [26] is an optimization algorithm based on the sequence model, which uses random forest to model the proxy function and has better exploration ability. SMAC will produce large variance for points with less exploration time and is good at dealing with optimization problems with many discrete values. For continuous temperature changes in the data center, the optimization effect is not good enough.

TPE is an improved version of nonstandard Bayes optimization algorithm based on tree structure Parzen, which is implemented in Sklearn's machine learning hyperparameter optimization toolkit Hyperopt [27]. The TPE algorithm constructs a probability model according to the past results and determines the next set of hyperparameters to be evaluated in the objective function by maximizing the expected improvement. Similar to the Bayes optimization method, the TPE algorithm can achieve good optimization performance, but it is also easy to fall into local optimum. Therefore, it is suitable for hyperparameter optimization problems in the low-dimensional space.

Because there are many random factors in the training of neural network, even if the same group of parameters are trained for many times, the results are not the same. In the previous methods, the program needs to be run many times, and the group with the best average performance is manually selected as the final result. We cannot obtain the optimal hyperparameters at one time [28, 29]. This paper proposes a hyperparameter optimization algorithm based on MLP

prediction. On the basis of Random Search, the MLP is used to predict the possible value of the hyperparameter space, and the neighbor value perturbation and stability model are used to improve the operation efficiency. The optimal hyperparameter with the best average accuracy and stability can be selected at one time.

## 3. Hyperparameter Optimization Algorithm Based on MLP

The hyperparameter optimization algorithm based on MLP is described in the following sections.

*3.1. The Overall Design and Process of the Algorithm.* Initially, the random search algorithm is adopted. Firstly, random sampling is carried out in the hyperparameter space and brought into the machine learning model to verify the effect. Then, some verified data are used to train the MLP model so as to predict the possible value of the hyperparameter space and take it as the basis for the next sampling.

Each time, we selected some hyperparameters which have a good verification effect and combined the idea of genetic algorithm to perturb the neighboring value to produce more high-quality solutions. The solutions predicted by MLP and obtained by disturbance are brought into the machine learning model again for verification. After several iterations, the validation data are gradually decreasing, the MLP model prediction accuracy is improved, and the hyperparameters optimization is finally realized.

Because of the uncertainty of the neural network training process, the algorithm uses the stability model in the final selection of the optimal hyperparameters. By calculating the square loss function of the model in the training process, and adding it with a certain weight coefficients to the objective function, it ensures that the finally selected hyperparameter has excellent performance and high stability.

The overall flow of the algorithm is shown in Figure 3. At the beginning, $N$ groups of hyperparameters are randomly sampled and substituted into the original machine learning model to verify its accuracy. Subsequently, the best performing $N = N/t$ group of hyperparameters is taken to train the MLP model until $n$ is reduced to 0. $t$ determines the descent speed of $N$. In this algorithm, $t$ is set to 2 and $N/k$ is the number of disturbances. In order to ensure the quality of hyperparameters generated by disturbances, the value of $k$ should not be too large. In this algorithm, $k = 10$ is set.

The pseudocode of the algorithm is shown in Algorithm 1.

The main advantages of the MLP hyperparameter optimization algorithm proposed in this study are as follows:

(1) use the MLP model to predict the possible results of the hyperparameter space and combine the gradient descent method to iterate the model to improve the model prediction accuracy.

(2) In the process of the algorithm, the neighboring value perturbation is added to mutate some high-quality solutions. The proportion of high-quality hyperparameter is increased and the search efficiency is improved.

(3) establish the loss function stability model of the training process. When the model is basically stable in the iteration process, we calculate the square sum of the RSME difference between each iteration and the last iteration of the hyperparameter and add it to the variance with a certain weight coefficients. In this way, the RSME and training smoothness can both be considered, and finally, a group of hyperparameter combinations with the best comprehensive performance can be selected.

*3.2. Multilayer Perceptron (MLP) Model Prediction Improves the Accuracy of Hyperparameter Sampling.* The MLP method is a multifunctional kind of ANN comprised of an input layer, hidden layers, and a single output layer. However, the MLP network with multilayers is more widely used, which solves the nonlinear problem that a single-layer perceptron cannot solve, and it has high accuracy. It has strong adaptive learning ability and is a highly parallel information processing system. Moreover, it does not depend on the mathematical model of the research object and has good robustness to the system parameter changes and external interference of the controlled object. Therefore, it is suitable for dealing with complex multiinput and multioutput nonlinear systems [30].

The MLP method can be expressed as

$$Y = F\left(\sum_{j=1}^{m} W_{kj}.F\left(\sum_{i=1}^{n} W_{ji}x_i + B_j\right) + B_k\right), \tag{9}$$

where $W_{kj}$ denotes the weight coefficients between the hidden layer and output layer, $W_{ji}$ defines the weight coefficients between the hidden layer and the input layer, $m$ is the hidden layer neurons number, $n$ is the number of neurons in the input layer, $B_j$ represents the offset of hidden layer neurons, $B_k$ represents the offset of neurons in the output layer, $F$ is the transfer function, and $Y$ is the output function. This study uses Sigmoid as the activation function, and the signal transfer functions for different values of $h$ are as follows:

$$\sigma(h) = \frac{1}{1 + \mathrm{EXP}(-h)}. \tag{10}$$

This algorithm first randomly samples in the hyperparameters space, brings it into the model to calculate the RSME corresponding to each group of hyperparameters, and generates a data set ($[\theta_1, \theta_2, \theta_3 \ldots \theta_n]$, RSME) for training the model of MLP. Take the hyperparameters and its RSME as input, the objective function is to minimize RSME. The verified sample hyperparameter combinations are divided into training set and test set according to the ratio of 0.8 and 0.2. The MLP model is trained to predict the entire hyperparameters space and guide the next sampling to avoid the blindness of random search. The predicted better results are brought into the model for verification again. After

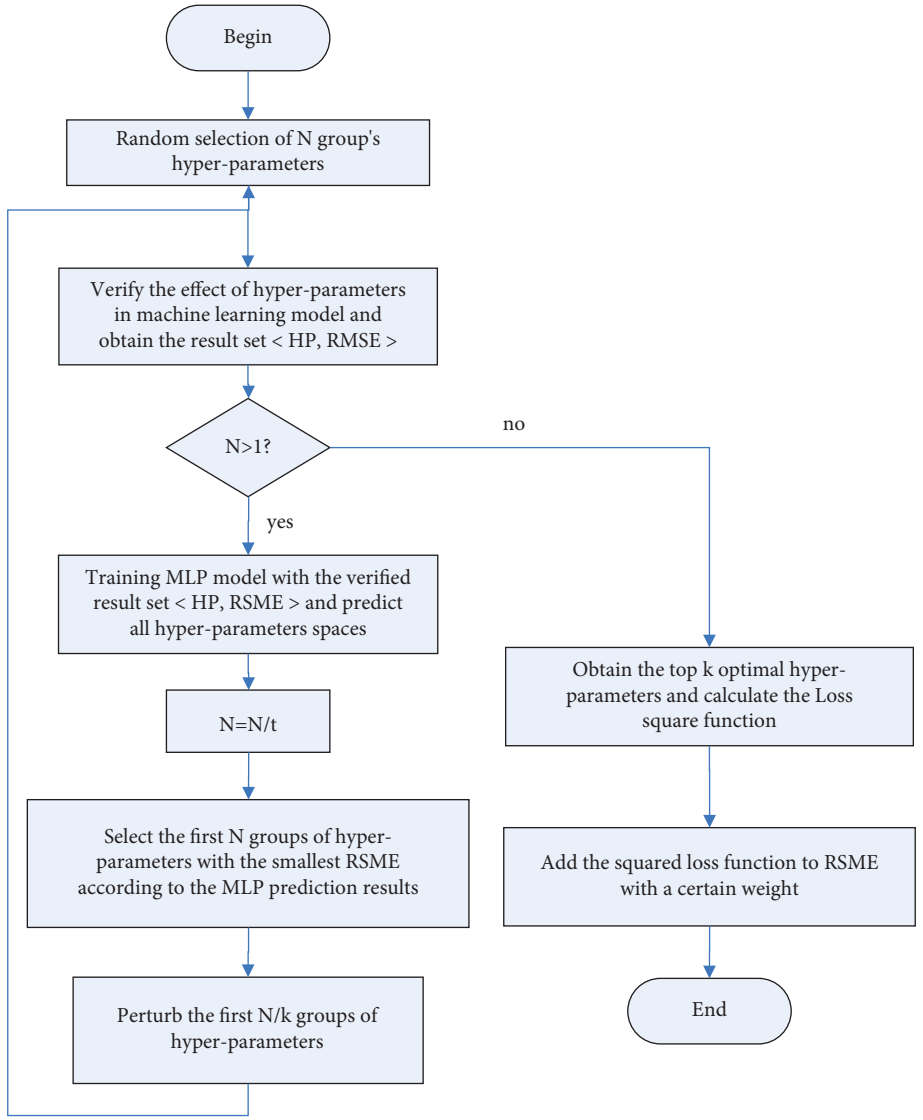


Figure 3: Flow chart of the hyperparameter optimization algorithm based on MLP.

several iterations, the accuracy of the model is continuously improved, and finally, the hyperparameters with excellent performance are obtained. Compared with Grid Search and Random Search, the hyperparameter sampling based on the MLP is more scientific and the optimization speed is faster.

*3.3. Neighboring Value Perturbation Promotes the Generation of High-Quality Hyperparameters.* After all the hyperparameters' space are predicted, some hyperparameter combinations which minimize the RSME of the objective function are selected to disturb the neighboring value to obtain more and better hyperparameters.

For example, for the hyperparameter combination $\{\theta_1, \theta_2, \theta_3 \ldots \theta_n\}$, we change one hyperparameter $\theta_i$ each time. If $\theta_i$'s in its domain is $k$, we select the close two values $\theta_{i[k-1]}, \theta_{i[k+1]}$ to replace parameter $\theta_i$. In general, changing one parameter will produce two new values $\theta_i'$ and $\theta_i''$. However, if the parameter itself is at the boundary of domain space, only a new hyperparameter combination $\theta_i'$ or $\theta_i''$ is generated:

$$\begin{cases} \theta_i' = \theta_{i[k-1]}, \text{if } \theta_{i[k-1]} \in Y_i, \\ \theta_i'' = \theta_{i[(k+1)]}, \text{if } \theta_{i[k+1]} \in Y_i, \end{cases} \tag{11}$$

where $\theta_{ik}$ represents the $k$th component of the $i$th hyperparameter in the hyperparameter space and $Y_i$ represents the domain of the $i$th hyperparameter. After a combination of hyperparameters is perturbed, $n$ to $2n$ new solutions are generated, as showed in the following:

$$\begin{cases} 1\text{: } (\theta_1', \theta_2 \ldots \theta_n), (\theta_1'', \theta_2 \ldots \theta_n), \\ 2\text{: } (\theta_1, \theta_2' \ldots \theta_n), (\theta_1, \theta_2'' \ldots \theta_n), \\ \ldots \\ i\text{: } (\theta_1 \ldots \theta_i', \ldots \theta_n), (\theta_1 \ldots \theta_i'' \ldots \theta_n), \\ \ldots \\ n\text{: } (\theta_1 \ldots \theta_{n-1}, \theta_n'), (\theta_1 \ldots \theta_{n-1} \ldots \theta_n''). \end{cases} \tag{12}$$

Each line represents to change a value of the hyperparameter space, and then replace the original hyperparameter value with two neighboring values. If the parent

**Input:** hyperparameter space ($X$), initial selection of hyperparameter amount ($N$), screening ratio ($t$), disturbance ratio ($k$), stability weight coefficients ($W$)
**Output:** a set of optimal hyperparameters
**Step:**
(1) $x$ = random_get_hyperparameters ($N$)//initially randomly select $N$ sets of hyperparameters from the hyperparameter space
(2) $R$ = run_and_get_loss ($A$ ($x$))//Train the $A$ model with the selected hyperparameters to get RMSE
(3) MLP.fit ($x$, $R$)//Training MLP model with trained hyperparameters and their RMSE
(4) While ($N > 1$) Do
(5) $N = N/t$//Gradually reduce $N$ according to the screening ratio
(6) MLP.predict ($x$)//Predict the RMSE of hyperparameters with MLP model
(7) $x$ = use_MLP_get_hyperparameters ($N$)//
Select the top $N$ sets of hyperparameters according to the MLP prediction results
(8) $x$ = ++use_disturbance_hyperparameters ($N/k$)//Perturb the optimal hyperparameters according to a certain proportion to increase the possibility of optimal solutions
(9) $R$ = run_and_get_loss ($A$ ($x$))
(10) MLP.refit ($x$, $R$)//retraining MLP
(11) For $i \leq k$ DO//Take the first $k$ groups of hyperparameters with the smallest RSME
(12) $L[i]$ = W∗Normalized_Quadratic loss function $[i]$ +RSME$[i]$//Normalize the squared loss function
(13) Return $x$, $R$, $L$

Algorithm 1: Hyperparameter optimization algorithm based on MLP.

individual value is at the boundary of the domain, only one new child hyperparameter will be generated, and the original one will be replaced with the new hyperparameter value. The operation is shown in (12).

The actual effect of each hyperparameter is verified in the model. Before verification, it is necessary to judge whether the hyperparameter combination exists in the verified data set. If it exists, this hyperparameter will be skipped directly. The results of all experimental individuals whose objective function (RSME) value is less than their parents are added to the training data. Otherwise, the experimental individuals will be eliminated and the parental individuals will be retained.

Table 1 shows the partial hyperparameter disturbance effects. The results show that, in each iteration, 89.6% of the optimal solution is improved by neighbor value perturbation, and the maximum improvement rate is 49.1%. By the disturbance of hyperparameter, the target function is reduced more effectively, and the optimization efficiency is effectively improved.

*3.4. Stability Model of the Training Process.* The loss and val_loss functions in the open source artificial neural network library Kera reflect the loss values of the training set and the test set. In the course of training, the more stable the loss function is, the more stable the performance of the model is.

The precision of several groups of hyperparameters finally selected by the optimization algorithm is often very close. The usual method is to select the best combination of parameters by running it multiple times. The time and calculation cost of this method is too high. In this algorithm, the loss function in the training process is modeled, the square loss function is calculated first, and then added them to RSME with a certain weight coefficients to ensure that the hyperparameter combination has better stability while maintaining high precision, which is given as follows:

$$L = W * \text{Normalized}_{\text{Quadratic}_{\text{Loss}_{\text{Function}[i]}}} + \text{RSME}[i]. \quad (13)$$

Normalized_Quadratic_Loss_Function is the square loss function of the $i$th group of hyperparameter, which is normalized. The weight coefficients of $W$ can be adjusted to set the proportion of stability. The value of $W$ shall not be too high to ensure the accuracy of hyperparameters. In this article, our weight coefficient $W = 0.002$. In order to verify the effectiveness of stability weighting calculation, we conducted the hyperparameter optimization experiment of LSTM algorithm. The performance of two groups of hyperparameters in the actual training process is selected, as shown in Figures 4 and 5. It can be seen from the figure that, with the increase of training times, the variance of loss function of the training set and test set gradually decreases and finally tends to be stable. The final prediction accuracy (RSME) of the two groups of hyperparameters is very close. The RSME of Figure 5 is 0.290, which is slightly better than that of Figure 4. However, from the stability of the training process, Figure 4 is obviously better than Figure 5. According to the traditional hyperparameter optimization algorithm, the hyperparameter in Figure 5 with the smallest RSME is selected. However, through the stability calculation of the two groups of hyperparameter, we found that the variance of the hyperparameter, shown in Figure 5, is only slightly higher than that in Figure 4, but the stability is much worse than that in Figure 4.

According to our algorithm, the optimal hyperparameter combination is ["sigmoid," 120.256.0, 'Adam] RSME = 0.291.

The experiment ran the two sets of hyperparameters three times, respectively, and the results are recorded in Table 2. It is shown that the hyperparameter combination ["

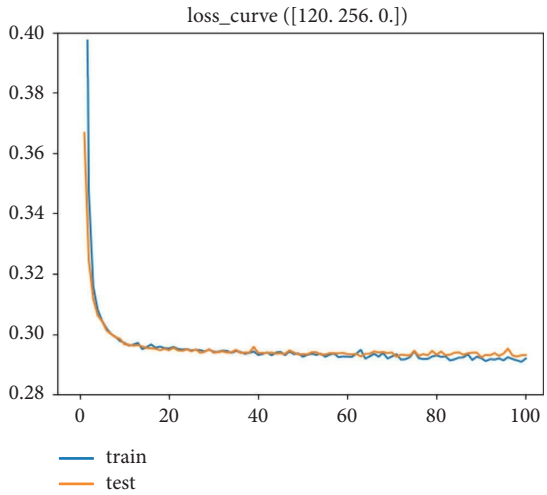


Figure 4: Effect of hyperparameter ["sigmoid", 120.256.0, "Adam] (RSME = 0.291, the single operation result is not the best, but the stability is high).

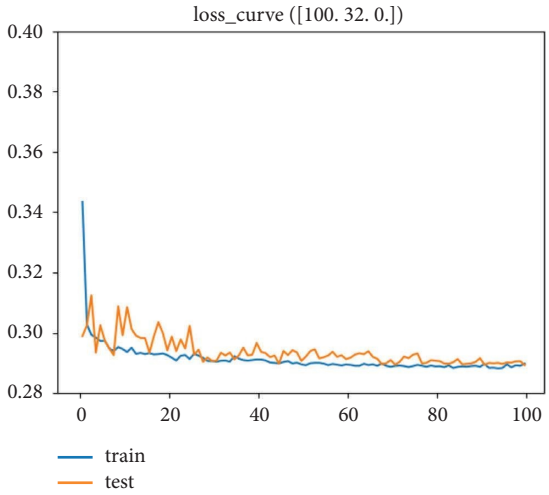


Figure 5: Effect of hyperparameter [sigmoid, 100.32.0, Ada] (RSME = 0.290, the single operation result is the best, but the stability is poor).

sigmoid," 120.256.0, "Adam"] selected by the algorithm proposed in this paper is not the lowest in a single run, but the average value of RSME is the lowest after multiple verifications. The traditional method needs to run many times to get the correct results, but our algorithm only needs to run once, which shows higher efficiency in practical application.

## 4. Experimental Results and Analysis

In order to optimize the effect and use the range of the algorithm, we conducted optimization experiments on two different machine learning models and jointly used prediction accuracy, stability, and operation efficiency indicators to evaluate the optimization effect.

Table 1: Part of the optimal value perturbation results.

| | Unit | Batch_Size | Dropout | RSME | Update (%) |
|---|---|---|---|---|---|
| Predictive | 190 | 4 | 0 | 13.72 | 49.1 |
| | 180 | 4 | 0 | 7.21 | |
| | 200 | 4 | 0 | 6.96 | |
| Disturb | 190 | 2 | 0 | 11.15 | |
| | 190 | 8 | 0 | 13.23 | |
| | 190 | 4 | 0.05 | 12.61 | |
| Predictive | 200 | 16 | 0 | 7.46 | 8.85 |
| | 190 | 16 | 0 | 6.80 | |
| | 200 | 8 | 0 | 12.48 | |
| Disturb | 200 | 32 | 0 | 12.17 | |
| | 200 | 16 | 0.05 | 10.76 | |
| | 200 | 16 | 0.05 | 10.76 | |

*4.1. Experimental Method and Data Set.* For data centers, temperature rise or fall is a gradual process related to time series. Therefore, we choose long short term memory (LSTM), which has great advantages in processing data highly correlated with time series for data center temperature prediction modeling. The model is trained by using the temperature monitoring data of the data center and the actual operating parameters of the server. It can accurately predict the evolution of the server inlet temperature under the dynamic load in the future. The LSTM temperature prediction model is a data center energy management algorithm developed by our project team, which has been applied in practice.

The experimental data use the temperature data set measured by the project team in the actual data center computer room, including 15 servers (model: Dell PowerEdge 850) and 2 air conditioners. The server is placed on a rack, and the air conditioner adopts the mode of air supply on the elevated floor and air return on the top. A wireless temperature sensor (model: telosb mote tpr2420ca) is placed at the inlet and outlet of each server. A temperature sensor and air velocity sensor (model: degree f333) are placed at the floor air inlet to monitor the air speed and the flow rate of the air conditioner. The temperature of the server using the IPMI protocol to collect.

A total of 5899 pieces of data were collected from the 25 hour operation data of the data center. It includes 46 parameters such as the air inlet and outlet temperature of the server, the temperature of the CRAC air outlet and air inlet, the indoor and outdoor temperature, the temperature of the server, and CPU utilization. The data time interval is 5 seconds, and the server load changes dynamically.

After the original data are preprocessed, the entire data set is divided into two parts: a training set and a test set. The training set accounts for 80% of the entire data and is used for the establishment of machine learning models. The remaining 20% is the test set and is used for measuring the performance of the selected hyperparameters.

For the universality of the algorithm, we verified it by another representative machine learning algorithm, i.e., Random Forest. Because in reference [31], by comparing the effectiveness of 179 machine learning algorithms, Random

Forest is proved to be a very excellent machine learning algorithm. We used the UCI standard data set (Beijing PM2.5 atmospheric data) to establish a Random Forest model. Similarly, the training set and verification set were established using the same proportions as above.

The corresponding hyperparameters and their ranges of the two machine learning algorithms are listed in Table 3.

*4.2. Experimental Results and Analysis.* At present, the most representative optimization algorithms include Random Search, Grid Search, Bayes Optimization, and its variants. Since Grid Search has been proved to be less effective than Random Search, no comparison will be made in this study. There are many variations of Bayes optimization, but the principle is basically the same. All of them are based on certain sampling, and the agent is established to carry out the next sampling. The algorithm proposed in this paper is also an improvement of Bayes method in principle. Therefore, we choose the most concise, efficient, and representative Gaussian Bayes algorithm for comparative experiments, which can start from one point to find the next better one.

In the experiment, two typical machine learning algorithms, LSTM and Random Forest, were modeled using the operation data of the own data center and the UCI standard data sets, and three kinds of hyperparameter optimization algorithms were compared.

*4.2.1. Hyperparameter Optimization Experiment of the Data Center LSTM Model.* Figure 6 shows the optimization effect of Random Search, Gaussian Bayes, and MLP algorithms on the temperature model of LSTM data center.

Since both the algorithm proposed in this paper and Gaussian Bayes need to accumulate a certain number of random sampling results, in order to facilitate comparison, the first 100 groups of hyperparameters of the two algorithms use the same initial random sampling (represented by black lines).

It can be seen from the figure that the random algorithm has no obvious change trend in the process of 200 samples. The better two group hyperparameters appear in the 6th (RSME = 0.316) and 47th (RSME = 0.315) part of the sampling. With continuous sampling, no better hyperparameters are found. However, for Gaussian Bayes and MLP algorithms, after 100 initial random samples, RSME gradually decreases. Gaussian Bayes finally converged at RSME = 0.314 (196th), and the MLP-based optimization algorithm achieved RSME = 0.291 (193rd). Among the three algorithms, our algorithm has the best optimization effect.

In order to show the training process more clearly, Figures 7 and 8 show the iterative process of MLP and Gaussian Bayes optimization, respectively. The vertical line in the figure indicates the beginning of a new round of iteration of MLP algorithm. We also made a vertical line at the corresponding position in the Gaussian Bayes iteration process diagram (Figure 8) to make a better comparison.

As can be seen from Figure 7, the whole training process has six iterations. In each iteration, the MLP model is used to predict the possible values of all hyperparameters spaces, and some effective hyperparameters are selected to repeat the training model. The training data selected each time are gradually reduced, and the prediction accuracy of the model is rapidly improved. In this way, the accuracy of the next sampling is effectively improved, and the optimal combination of hyperparameter is finally predicted.

GPR selects one group of hyperparameter to train and update the model according to the prediction results each time. In the initial stage of iteration, RSME has an obvious rising process, as shown in Figure 8. The highest RSME in the iterative process reaches 0.989, even exceeding the highest RSME in the initial sampling stage, which is 0.973, and then gradually decreases. After trying about 100 sets of hyperparameter, the result becomes stable. The newly selected hyperparameters are not changing, and the appropriate hyperparameters can be considered to be found. But in fact, this group of hyperparameters is ["sigmoid," 0.2, 64, 110, "Adam"], and its RMSE value is about 0.32, which is not the optimal result. Due to the limitations of fitting, Bayes optimization can easily fall into a local optimal solution. In the actual use of Bayes optimization, we often set an acceptable threshold of the accuracy of the results, that is, to find the hyperparameters within a certain accuracy range.

*4.2.2. Hyperparameter Optimization Experiment of the Random Forest Model.* Figure 9 shows the optimization effects of three kinds of hyperparameter optimization algorithms on the Random Forest model. The data set is from the Beijing PM2.5 atmospheric data (prsa_data_2010.1.1–2014.12.31. CSV) of UCI standard data set. The hyperparameters to be optimized and their value ranges are shown in Table 3.

As showed in the figure, the Random algorithm tries 200 random samples, and MLP and Gaussian Bayes algorithms use the same results of the first 100 samples. Among the three algorithms, the MLP algorithm also shows the best optimization effect on the RF model, with RSME = 0.0562. The second is random algorithm, whose hyperparameter RSME = 0.0566, and then Gaussian Bayes finally converges to RSME = 0.0571. Gaussian Bayes and the MLP-based algorithm, after 100 random samples, and the subsequent samples are gradually accurate, then RSME decreases and finally converges. Although Random Search has achieved lower RSME than Gauss, there are also more samples with poor accuracy. This is due to its strong randomness and lack of effective evidence. From the above experiments, we can see that our algorithm can be extended to more machine learning models and is still effective.

In terms of running speed, the three algorithms need a lot of sampling and verification, and the operation time is long. The running time mainly depends on the calculation speed of the original machine learning model. Random Search, Gaussian Bayes, and MLP algorithms take 1379 s, 1703 s, and 1406 s to optimize the LSTM model, respectively, while the time to optimize the RF model is only 356 s, 409 s, and 370 s. Since the RF operation speed is higher than that of LSTM, the difference in operation time is greater after hundreds of sampling. For the same machine learning

Table 2: The results of three runs of hyperparameter combinations.

| Run number | Hyperparameter | |
|---|---|---|
| | ["sigmoid," 120.256.0, "Adam"] | ["sigmoid," 100.32.0, "Adam"] |
| 1 | 0.291 | 0.290 |
| 2 | 0.293 | 0.295 |
| 3 | 0.291 | 0.293 |
| Average | 0.2917 | 0.2927 |

Table 3: Algorithm model and corresponding hyperparameters.

| Algorithm | Hyperparameter | Ranges | Interval |
|---|---|---|---|
| LSTM | Act.ivation | ["linear," "sigmoid", "relu," "tanh"] | 0.1 |
| | Dropout | [0~0.5] | 10 |
| | Unit | [20~200] | $N$ *2 |
| | Batch_Size | [32, 64~512] | |
| | Optimizer | ["SGD," "adagrad," "Adadelta," "Adam," "nAdam"] | |
| Random forest | n-estimators max_depth | [100~1200] | 100 |
| | min_samples_split | [2~30] | 2 |
| | min_samples_leaf | [1~99] | 2 |
| | max_features | [1~9] | |
| | criterion | [sqrt, log2, None] | |
| | bootstrap | [gini, entropy] [True, False] | 2 |

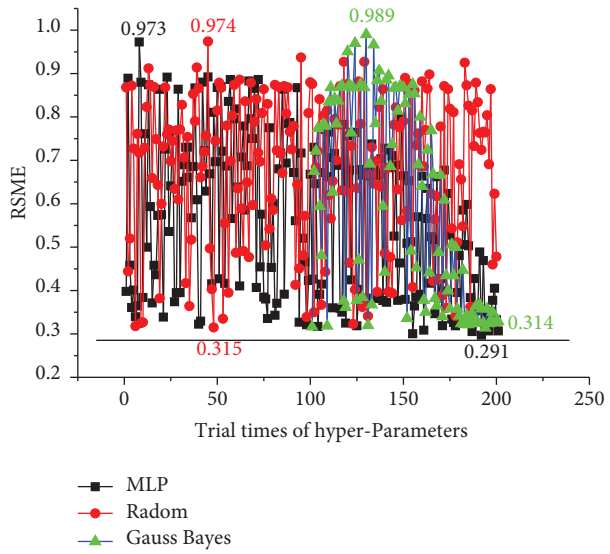


Figure 6: The hyperparameter optimization process of the LSTM model (the horizontal axis represents the number of training sessions; the vertical axis represents the accuracy of the selected hyperparameter on the validation set after each sampling. MLP stands for the algorithm proposed in this paper; Random stands for Random Search; Gauss Bayes stands for Gaussian Bayes algorithm).

model, Random Search runs the fastest because of its simple sampling method. MLP only runs at the beginning of each iteration, predicting all the hyperparameter values, and runs a total of six times in the whole iteration process. However, Gaussian Bayes needs to run every sampling, so MLP runs faster than Gaussian Bayes. In general, all the three algorithms take a lot of time, so they are not suitable for online optimization.

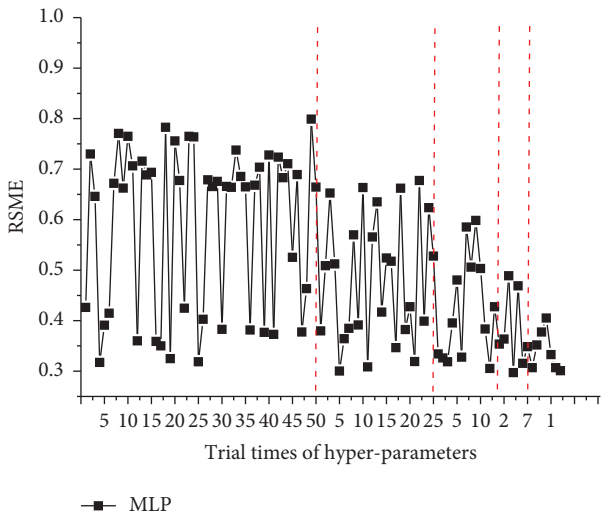


Figure 7: MLP Model iteration process.

*4.2.3. Conclusion.* Through the above two groups of experiments, it can be seen that the hyperparameter optimization algorithm based on MLP proposed in this paper have better results than the Random Search and Gaussian Bayes algorithm. With each iteration, the MLP model will become moreaccurate, and the proportion of training parameter groups will become less accurate, and the training hyperparameter results will become better. Although Random Search may find better results, it will lead to great uncertainty due to the randomness of the sampling process. Especially with the increase of search space, the search efficiency of Random Search algorithm decreases, and the time spent on model training will increase.

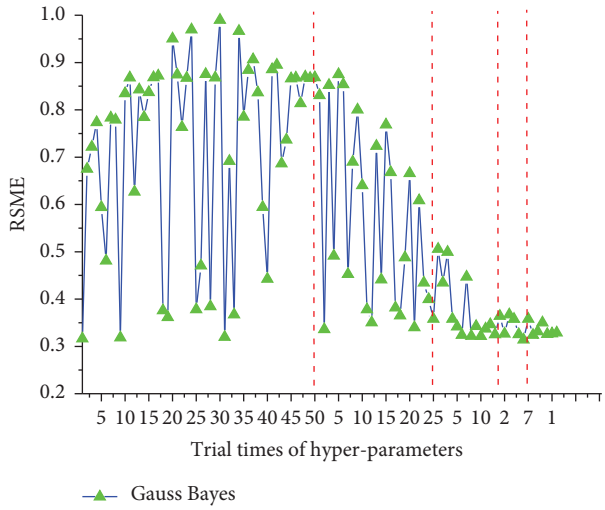


Figure 8: Bayes hyperparameter iteration process.

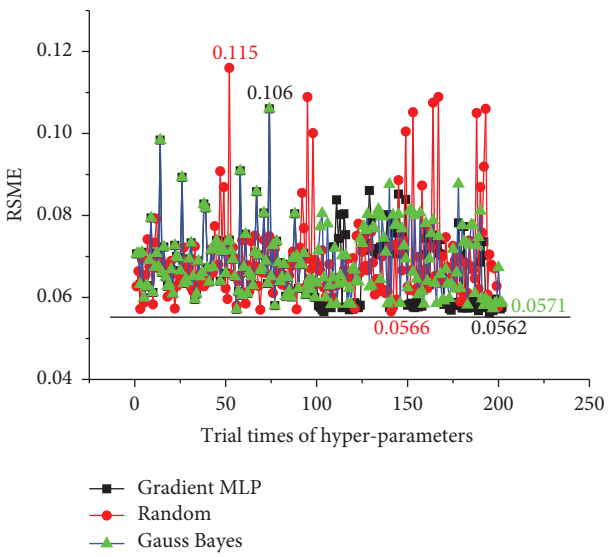


Figure 9: Hyperparameter optimization of the Random Forest mode.

The Gaussian Bayes method can predict the posterior distribution of the data set according to only a few data and gradually settles the optimal hyperparameters from one point. The disadvantage is the great dependence on the initial point, which is easy to cause overfitting and eventually falls into the local optimum. In general, Bayes optimization usually sets an accuracy range, and when hyperparameters with acceptable accuracy are found, Bayes Optimization stops optimization.

In fact, the hyperparameter optimization algorithm in this paper cannot guarantee that the optimal hyperparameters will be selected at the end of the iteration. At present, there is no hyperparameter optimization algorithm to ensure that the optimal hyperparameter can be obtained because there are random factors in the training process of neural networks. However, during each iteration of our algorithm, the performance of the selected hyperparameter is getting better and better.

In the algorithm, we use the idea of mutation in the genetic algorithm to perturb the neighbor value of high-quality solution. In Section 3.3, it has been proved that 89.6% of the perturbations have improved the original hyperparameters, and the variation accuracy of some hyperparameters is greatly improved. When the number of attempts is the same, the proposed MLP based hyperparameters optimization algorithm has better results and faster speed than the Gaussian Bayesian optimization algorithm.

Hyperparameter optimization algorithms usually require a long training time. It can be seen from the above experiment that the optimization for LSTM needs more than 1300 seconds, and the optimization for Random Forest also needs more than 300 seconds, which takes a long time. It will be highly inefficient to obtain the hyperparameters through the traditional multiple running procedures and manual selection results. By establishing the stability model of the loss function in the training process, the algorithm only needs to run once to obtain high precision and high stability hyperparameter. It has the same effect as other algorithms by calculating the average value through multiple runs, which greatly improves the efficiency of the algorithm. But even so, this algorithm is still only applicable to general off-line hyperparameter optimization scenarios.

## 5. Summary and Outlook

The temperature of the server changes much faster than the ambient temperature. Understanding the temperature changes in the data center and adjusting the refrigeration equipment in advance can effectively avoid temperature lag and hot spots. The machine learning algorithm cannot rely on the specific machine room structure, and only uses historical data to model the data center. It can predict the temperature change of the data center according to the upcoming tasks, with the characteristics of rapid prediction and early sensing of temperature change. However, the accuracy of machine learning algorithm is greatly affected by the hyperparameters.

The hyperparameter optimization algorithm of LSTM temperature prediction model proposed in this paper is realized through four steps: (1) random sampling; (2) establish the MLP model according to the adopted results; (3) perturb part of the prediction results, and repeat the training model with effective prediction values and perturbation values; and (4) evaluate the stability of the hyperparameters in the training process, and select the optimal hyperparameter. The method of neighbor value perturbation improves the probability of optimal solution; the method of stability evaluation changes the traditional method of calculating the average value through multiple operations. It can select the optimal hyperparameter at one time, greatly improving the efficiency.

In practical application, this algorithm optimizes the LSTM temperature prediction model of our data center and effectively improves the prediction accuracy of the model. Based on the prediction data, we achieved accurate refrigeration regulation and server task scheduling. In this way,

the safety of the server is ensured, and the long-term low-temperature operation of the refrigeration equipment is avoided, and the energy saving of the data center is effectively realized. This algorithm is not only applicable to LSTM model. We have carried out experiments on two machine learning models, random forest model, and LSTM, and compared them with the Gaussian Bayes method and Random Search. The results show that our algorithm is equally effective for the two machine learning algorithms and has good performance in accuracy and efficiency, which proves the availability for other machine learning methods.

*5.1. Suggestions for Improvement.* In each training process of the MLP model, we will adjust the amount of hyperparameters to take into account the optimization speed. At present, we set the number of hyperparameters we tried at each iteration to half of the previous one. In the case of poor initial random sampling, this method may still not find the optimal solution when the iteration is completed. In future research, we can make the number of hyperparameters tried in each iteration set automatically according to the effect. By increasing the number of attempts in the case of poor random sampling and reducing iterations in the case of good random sampling, the optimization effect is guaranteed.

## Data Availability

The data used to support the findings of the study were obtained from high performance computing center of Xi'an Jiaotong University.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

This work was supported by National Key Research and Development Plan of China (no. 2017YFB1001701).

## References

[1] C. D. Patel, C. E. Bash, and A. H. Beitelmal, *Smart Cooling of Data Centers*, American Society of Mechanical Engineers (ASME), New York, NY, USA, 2003.

[2] J. Kim, M. El-Khamy, and J. Lee, "Residual LSTM: design of a deep recurrent architecture for distant speech recognition," *Interspeech*, vol. 12, pp. 1591–1595, 2017.

[3] T. N. Sainath, R. J. Weiss, K. W. Wilson et al., "Multichannel signal processing with deep neural networks for automatic speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 5, pp. 965–979, 2017.

[4] U. Singh, A. Singh, S. Parvez, and S. Amip, "Cfd-Based Operational Thermal Efficiency Improvement of a Production Data Center," in *Proceedings of the 1st USENIX Conference on Sustainable Information Technology*, pp. 17–24, San Jose CA, Feburary 2010.

[5] J. Chen, R. Tan, and Y. Wang, "A high-fidelity temperature distribution forecasting system for data centers," in *Proceedings of the 33rd IEEE Real-Time Systems Symposium*, pp. 215–224, NW Washington, DC, USA, December 2012.

[6] C. H. Lin and S. Lucey, *Inverse Compositional Spatial Transformer Networks*, pp. 2252–2260, IEEE Computer Society, Washington, D.C., USA, 2017.

[7] R. Girshick, J. Donahue, and T. Darrell, *Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation*, IEEE Computer Society, Washington, D.C., USA, 2013.

[8] J. Moore, J. S. Chase, and R. P. Weatherman, "Automated, online and predictive thermal mapping and management for data centers," in *Proceedings of the IEEE International Conference on Autonomic Computing*, pp. 155–164, Umeå, Sweden, May 2006.

[9] L. Li, C. J. M. Liang, and J. Liu, "Thermocast: a cyber-physical forecasting model for datacenters," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1370–1378, New York, NY, USA, Auguest 2011.

[10] Y. X. Xu, W. G. Wu, S. M. Wang, Z. Hu, and S. Cui, "Data center temperature prediction algorithm based on Long Short-Term Memory network," *Computer Technology and Development*, vol. 29, no. 12, pp. 1–7, 2019.

[11] J. Wu, S. P. Chen, X. Y. Chen, and R. Zhou, "Reinforcement learning for model selection and hyper-parameter optimization," *Journal of University of Electronic Science and Technology of China*, vol. 49, no. 02, pp. 255–261, 2020.

[12] H. Wang, T. Tao, and X. Mei, G. Hongfang, Experimental study on condensation heat transfer characteristics inside an inclined wave-finned flat tube of direct air-cooling system," *Journal of Thermal Science*, vol. 30, pp. 1–9, 2020.

[13] Y. Guo, J. Y. Li, and Z. H. Zhan, "Efficient hyper-parameter optimization for convolution neural networks in deep learning: a distributed particle swarm optimization approach," *Cybernetics & Systems*, vol. 52, no. 1, pp. 36–57, 2020.

[14] H. Wang, T. Tao, J. Xu, X. Mei, X. Liu, and P. Gou, "Cooling capacity of a novel modular liquid-cooled battery thermal management system for cylindrical lithium ion batteries," *Applied Thermal Engineering*, vol. 178, Article ID 115591, 2020.

[15] Q. Tao, F. Liu, Y. Li, and D. Sidorov, "Air pollution forecasting using a deep learning model based on 1D convnets and bidirectional GRU," *IEEE Access*, vol. 7, pp. 76690–76698, 2019.

[16] A. V. Zhukov, D. N. Sidorov, and A. M. Foley, "Random forest based approach for concept drift handling," in *Proceedings of the International Conference on Analysis of Images, Social Networks and Texts*, Springer, Cham, Yekaterinburg, Russia, April 2016.

[17] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," *Neural Computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[18] R. W. Liu, M. Liang, J. Nie, W. Y. B. Lim, Y. Zhang, and M. Guizani, "Deep learning-powered vessel trajectory prediction for improving smart traffic services in maritime internet of things," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 5, pp. 3080–3094, 2022.

[19] S. P. Chen, J. Wu, and X. Y. Chen, "Hyper-parameter optimization method based on reinforcement learning [J]," *Journal of Chinese Computer Systemsn*, vol. 41, no. 04, pp. 679–684, 2020.

[20] S. Katakami, H. Sakamoto, and M. Okada, "Bayesian hyperparameter estimation using Gaussian process and bayesian optimization," *Journal of the Physical Society of Japan*, vol. 88, no. 7, Article ID 074001, 7 pages,, 2019.

[21] A. Akl, I. El-Henawy, and A. Salah, "Optimizing deep neural networks hyperparameter positions and values," *Journal of Intelligent and Fuzzy Systems*, vol. 5, no. 3, pp. 1–17, 2019.
[22] J. Bergstra, "Bengio Y Random search for hyper-parameter optimiza-tion," *Journal of M achine Learning Research*, vol. 13, no. 1, pp. 281–305, 2012.
[23] J. Snoek, H. Larochelle, and R. Adams, "Practical bayesian optimization of machine learning algorithms," in *Proceedings of the Neural Infor-mation Processing Systems*, pp. 2951–2959, Montreal, Canada, December 2012.
[24] T. T. Joy, S. Rana, and S. Gupta, "Fast Hyper-Parameter Tuning Using Bayesian Optimization with Directional derivatives," *Knowledge-Based Systems*, vol. 205, Article ID 106247, 2020.
[25] A. Klein, S. Falkner, N. Mansur, and B. Ro, "A flexible and robust bayesian optimization framew ork in python," in *Proceedings of the Neural Information Processing Systems Workshop on Bayesian Op-Timization*, pp. 1–5, Xiamen, China, April 2017.
[26] H. Frank, H. H. Holger, and L. Kevin, "Sequential model-based optimization for general algorithm configuration," *Learning and Intelligent Optimization*, vol. 5, no. 8, pp. 507–523, 2011.
[27] J. Bergstra, R. Bardenet, and Y. Bengio, "Algorithms for hyperpa-rameter optimization," in *Proceedings of the Neural Information Pro-cessing Systems*, pp. 2546–2554, Long Beach CA, USA, October 2011.
[28] L. Marius and H. Frank, "Warmstarting of Model-Based Algo-Rithm Configuration," in *Proceedings of the Association for Advance-ment of Artificial Intelligence*, pp. 1355–1362, Burlingame, CA, USA, April 2018.
[29] Y. Q. Hu and H. Qian, "Sequential Classification-Based Op-Timization for Direct Policy searc," in *Proceedings of the Association for Advancement of Artificial Intelligence*, pp. 2029–2035, San Francisco CA, USA, May 2017.
[30] S. Shadkani, A. Abbaspour, S. Samadianfard, S. Hashemi, A. Mosavi, and S. S. Band, "Comparative study of multilayer perceptron-stochastic gradient descent and gradient boosted trees for predicting daily suspended sediment load: the case study of the Mississippi River, U.S.A," *International Journal of Sediment Research*, vol. 36, no. 4, pp. 512–523, 2021.
[31] E. Cernadas and D. Amorim, "Do we need hundreds of classifiers to solve real world lassification problemsn?" *The Journal Journal of M Achine Learn-Ing Research*, vol. 15, no. 1, pp. 3133–3181, 2014.