Digital Object Identifier:10.11989/JEST.1674-862X.80904120

Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization

Jia Wu* | Xiu-Yun Chen | Hao Zhang | Li-Dong Xiong | Hang Lei | Si-Hao Deng

***Abstract*—**Hyperparameters are important for machine learning algorithms since they directly control the behaviors of training algorithms and have a significant effect on the performance of machine learning models. Several techniques have been developed and successfully applied for certain application domains. However, this work demands professional knowledge and expert experience. And sometimes it has to resort to the brute-force search. Therefore, if an efficient hyperparameter optimization algorithm can be developed to optimize any given machine learning method, it will greatly improve the efficiency of machine learning. In this paper, we consider building the relationship between the performance of the machine learning models and their hyperparameters by Gaussian processes. In this way, the hyperparameter tuning problem can be abstracted as an optimization problem and Bayesian optimization is used to solve the problem. Bayesian optimization is based on the Bayesian theorem. It sets a prior over the optimization function and gathers the information from the previous sample to update the posterior of the optimization function. A utility function selects the next sample point to maximize the optimization function. Several experiments were conducted on standard test datasets. Experiment results show that the proposed method can find the best hyperparameters for the widely used machine learning models, such as the random forest algorithm and the neural networks, even multi-grained cascade forest under the consideration of time cost.

***Index Terms*—**Bayesian optimization, Gaussian process, hyperparameter optimization, machine learning.

1. Introduction

Normally machine learning algorithm transforms a problem that needs to be solved into an optimization problem and uses different optimization methods to solve the problem. The optimization function is composed of multiple hyperparameters that are set prior to the learning process and affect how the machine learning

*Corresponding author

Manuscript received 2018-07-03; revised 2018-08-09.

This work was supported in part by the National Natural Science Foundation of China under Grant No. 61503059.

J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, and H. Lei are with the School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China (e-mail: jiawu@uestc.edu.cn; 996167678@qq.com; 857221751@qq.com; 81912416@qq.com; hlei@uestc.edu.cn).

S.-H. Deng is with Université de Technologie de Belfort-Montbéliard, Belfort 90010, France (e-mail: sihao.deng@utbm.fr).

Color versions of one or more of the figures in this paper are available online at http://www.journal.uestc.edu.cn.

Publishing editor: Xin Huang

algorithm fits the model to data. Hyperparameters are different from the internal model parameters, such as the neural network's weights, which can be learned from the data during the model training phase. Before the training phase, we would like to find a set of hyperparameter values which archive the best performance on the data in a reasonable amount of time. This process is called hyperparameter optimization or tuning. It plays a vital role in the prediction accuracy of machine learning algorithms. Unfortunately, the relationship between the performance of the machine learning algorithms and hyperparameters is unclear. In practice, it is necessary to continuously adjust hyperparameters and train a number of models with different combinations of values, and then compare the model performance to choose the best model. Therefore, how to optimize hyperparameters becomes a key issue in a machine learning algorithm.

There are mainly two kinds of hyperparameter optimization methods, i.e., manual search and automatic search methods. Manual search tries out hyperparameter sets by hand. It depends on the fundamental intuition and experience of expert users who can identify the important parameters that have a greater impact on the results and then determine the relationship between certain parameters and final results through the visualization tools[1]. Manual search requires users to have more professional background knowledge and practical experience. And it is hard to be applied by non-expert users. The process of tuning hyperparameters is not easily reproducible. Besides, as the number of hyperparameters and the range of values increase, it becomes quite difficult to manage since humans are not good at handling high dimensional data and easily misinterpret or miss trends and relationships in hyperparameters.

To overcome the drawbacks of manual search, automatic search algorithms have been proposed, such as grid search[2] or Cartesian hyperparameter search. The principle of grid search is exhaustive searching. Grid search trains a machine learning model with each combination of possible values of hyperparameters on the training set and evaluates the performance according to a predefined metric on a cross validation set. Finally, grid search outputs hyperparameters that achieve the best performance. Although this method achieves automatic tuning and can theoretically obtain the global optimal value of the optimization objective function, it suffers from the curse of dimensionality, i.e., the efficiency of the algorithm decreases rapidly as the number of hyperparameters being tuned and the range of values of hyperparameters increase.

To solve the problem of expensive cost in grid search, the random search algorithm[2] has been proposed, which found that for most data sets, only a few of the hyperparameters really matter. The overall efficiency can be improved by reducing the search to hyperparameters that do not matter, and finally the approximate solution of the optimization function is obtained. Random search tries random combinations of a range of values. Compared with grid search[3], random search is more efficient in a high-dimensional space. However, [2] shows that random search is unreliable for training some complex models.

Therefore, how to make the automatic tuning algorithm achieve high precision and high efficiency has always been a problem that has not yet been fully solved in machine learning.

Hyperparameter tuning is an optimization problem where the objective function of optimization is unknown or a black-box function. Traditional optimization techniques like Newton method or gradient descent cannot be applied. Bayesian optimization is a very effective optimization algorithm in solving this kind of optimization problem[4]. It combines prior information about the unknown function with sample information, to obtain posterior information of the function distribution by using Bayesian formula. Then, based on this posterior information, we can deduce where the function obtains the optimal value. Experimental results show that Bayesian optimization algorithm outperforms other global optimization algorithms[5]. Therefore, we apply Bayesian optimization based on Gaussian process to tune hyperparameters of machine learning. We assume that the optimization function obeys Gaussian distribution, then the prior distribution of hyperparameters can be determined. In the following sections, Bayesian

optimization algorithm based on Gaussian process will be described in detail at first. Then, experiments are carried out to verify the proposed method for three typical machine learning models.

This paper is organized as follows. The basic principle of Bayesian optimization is introduced in Section 2. Section 3 illustrates two key factors in Bayesian optimization algorithm: The update of the posterior distribution of the function and the selection of the acquisition function. In Section 4, Bayesian optimization is applied to tune hyperparameters for the most commonly used machine learning models, such as random forest, deep neural network, and deep forest. Experiments are conducted on standard datasets. Conclusion is drawn in the final section.

2. Overview of Bayesian Optimization

Bayesian optimization is an effective method for solving functions that are computationally expensive to find the extrema[6]. It can be applied for solving a function which does not have a closed-form expression. It can also be used for functions which are expensive to calculate, the derivatives are hard to evaluate, or the function is non-convex. In this paper, the optimization goal is to find the maximum value at the sampling point for an unknown function f:

$$x^+ = \arg\max_{x \in A} f(x). \tag{1}$$

where A denotes the search space of x. Bayesian optimization derives from Bayes' theorem[6],[7], i.e., given evidence data E, the posterior probability $P(M|E)$ of a model M is proportional to the likelihood $P(E|M)$ of overserving E given model M multiplied by the prior probability of $P(M)$:

$$P(M|E) \propto P(E|M)P(M). \tag{2}$$

The above formula reflects the core idea of Bayesian optimization. The principle of Bayesian optimization is to combine the prior distribution of the function $f(x)$ with the sample information (evidence) to obtain the posterior of the function; then the posterior information is used to find where the function $f(x)$ is maximized according to a criterion. The criterion is represented by a utility function u which is also called acquisition function. The function u is used to determine the next sample point in order to maximize the expected utility. When searching the sampling area, it is necessary to take into account both exploration (sampling from the areas of high uncertainty) and exploitation (sampling from that with high values[8]). That will help to reduce the number of sampling. Moreover, the performance will be improved even when the function has multiple local maxima.

In addition to the sample information, Bayesian optimization depends on the prior distribution of the function f, which is an indispensable element in the statistical inference of the posterior distribution of the function f. A priori distribution does not have to be an objective basis, either partially or entirely based on subjective beliefs. It is generally assumed that Gaussian process is well suited to the prior distribution of Bayesian optimization. Gaussian process is highly flexible and easy to handle, so Bayesian optimization applies Gaussian process to fit data and update the posterior distribution.

The Bayesian optimization algorithm is shown in Table 1, where $D_{1:t-1} = \{x_n, y_n\}_{n=1}^{t-1}$ represents the training dataset which consists of $t-1$ observations of function f. From the description of the algorithm, we can clearly see that the whole algorithm is composed of two parts: Updating the posterior distribution (steps 3

Table 1: Algorithm 1

Algorithm 1: Bayesian optimization
1: For t=1, 2, ⋯
2: Find x_t by optimizing the acquisition function u over function f: $x_t = \arg\max_x u(x\|D_{1:t-1})$.
3: Sample the objective function: $y_t = f(x_t)$.
4: Augment the data $D_{1:t} = \{D_{1:t-1}, (x_t, y_t)\}$ and update the posterior of function f.
5: End for.

and 4) and maximizing the acquisition function (step 2). As the observations accumulate, the posterior distribution is updated continuously; on the basis of the new posterior, the point where the acquisition function is maximized is found and added to the training dataset. The whole process is repeated until the maximum number of iterations is reached or the difference between the current value and the optimal value obtained so far is less than a predefined threshold. It is noted that Bayesian optimization does not require the explicit expression of function f compared with other optimization methods, such as gradient descent algorithms, and therefore it has a wider range of applications.

3. Bayesian Optimization

3.1. Gaussian Process

In the field of machine learning, Gaussian process is a kind of technique developed on the basis of Gaussian stochastic process and Bayesian learning theory. Gaussian process is a generalization of the Gaussian probability distribution. Whereas a probability distribution describes random variables which are scalars or vectors (for multivariate distributions), a stochastic process governs the properties of functions[9]. Gaussian process is a stochastic process such that any finite sub-collection of random variables has a multivariate Gaussian distribution. Gaussian process assumes that similar input produces similar output, and thus assumes a statistical model of the function. Like Gaussian distribution defined by mean and covariance, Gaussian process is defined by its mean function $m: x \to \mathbb{R}$ and its covariance function $k: x \times x \to \mathbb{R}$. We denote Gaussian process as

$$f(x) \sim \mathrm{GP}(m(x), k(x, x')). \tag{3}$$

Gaussian process is different from the Gaussian distribution, i.e. the probability density function $f(x)$ for an arbitrary x is no longer a scalar but a normal distribution function over all the possible values of $f(x)$. For convenience, assume the Gaussian process mean function $m(x)$=0. For the covariance function k, the exponential square function is a popular choice:

$$k(x_i, x_j) = \exp\left(-\frac{1}{2}\|x_i - x_j\|^2\right) \tag{4}$$

where x_i and x_j represent the ith and jth samples, respectively. When x_i and x_j get close together, the value of $k(x_i, x_j)$ approaches 1; otherwise, it approaches 0. It is noted that when two sampling points are close to each other, they have a strong correlation and a mutual influence; when they get further apart, the mutual influence is weak.

The process of determining the posterior distribution of $f(x)$ is as follows.

1) At first, sample t observations as the training set $D_{1:t} = \{x_n, f_n\}_{n=1}^{t}$, $f_n = f(x_n)$. Suppose the function values f are drawn according to a multivariate normal distribution $f \sim N(0, \mathbf{K})$, where

$$\mathbf{K} = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_t) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_t) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_t, x_1) & k(x_t, x_2) & \cdots & k(x_t, x_t) \end{bmatrix}. \tag{5}$$

Each element inside $\mathbf{K}$ is calculated by (4). The function k measures the degree of approximation between two samples. Obviously, the diagonal element $k(x_i, x_i) = 1$ without considering the effect of noise.

2) Based on the function f, compute the function value $f_{t+1} = f(x_{t+1})$ at the new sample point x_{t+1}. According to the assumption of Gaussian process, $\mathbf{f}_{1:t}$ in the training set plus the function value f_{t+1} follows the t+1 dimensional normal distribution:

$$\begin{bmatrix} \mathbf{f}_{1:t} \\ f_{t+1} \end{bmatrix} \sim N\left(0, \begin{bmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^{\mathrm{T}} & k(x_{t+1}, x_{t+1}) \end{bmatrix}\right) \tag{6}$$

where $\mathbf{f}_{1:t} = [f_1, f_2, \cdots, f_t]^{\mathrm{T}}$,

$$\mathbf{k} = [k(x_{t+1}, x_1)\, k(x_{t+1}, x_2) \cdots k(x_{t+1}, x_t)] \tag{7}$$

and f_{t+1} follows one-dimensional normal distribution, i.e. $f_{t+1} \sim N(\mu_{t+1}, \sigma_{t+1}^2)$. By the properties of the joint Gaussian (normal distribution),

$$\mu_{t+1}(x_{t+1}) = \mathbf{k}^{\mathrm{T}} \mathbf{K}^{-1} \mathbf{f}_{1:t} \tag{8a}$$

$$\sigma_{t+1}^2(x_{t+1}) = -\mathbf{k}^{\mathrm{T}} \mathbf{K}^{-1} \mathbf{k} + k(x_{t+1}, x_{t+1}). \tag{8b}$$

It can be seen from the above equations that Gaussian process does not return a scalar value for f_{t+1}, it returns the probability distribution over all possible values of f_{t+1} (as shown in Fig. 1). If the training set is large enough, Gaussian process can obtain an approximate estimate of the function $f(x)$ distribution. An intuitive representation of Gaussian process is shown in Fig. 1.

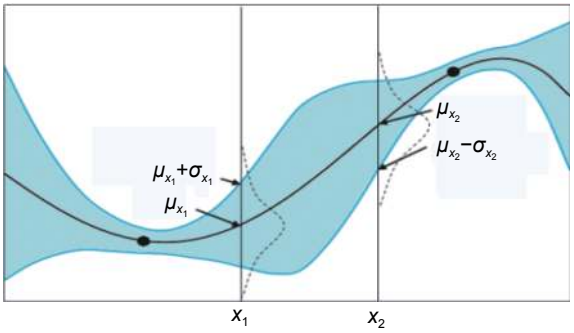


Fig. 1. One-dimensional Gaussian process.

Fig. 1 shows a one-dimensional Gaussian process with two observations. The black dots indicate the observations of data points. The black curve is the predicted mean of the objective function. The blue area represents the range of one standard deviation near the mean. As shown in Fig. 1, the variance is small when the function f is close to the observation; the variance is large while it is away from the observation.

3.2. Acquisition Function

After obtaining the posterior distribution of the objective function, Bayesian optimization uses the acquisition function u to derive the maximum of the function f. Normally, we assume that the high value of the acquisition function corresponds to the large value of the objective function f. Hence, maximizing the acquisition function is equivalent to maximizing the function f:

$$x^+ = \arg\max_{x \in A} u(x|D). \tag{9}$$

In the following, we will briefly introduce several common acquisition functions. Before the introduction, the definitions of some notations are given as follows: $\Phi(\cdot)$: Cumulative distribution function of the standard normal distribution; $\varphi(\cdot)$: Probability density function of the standard normal distribution; $x^+ = \arg\max_{x_i \in x_{1:t}} f(x_i)$, x^+ represents the position where the function f is maximized after obtaining t sample points.

1) Probability of improvement (PI)

Function PI tries to explore near the current optimal value point in order to find the points most likely to prevail over the current optimal value. The search process continues until the number of iterations of the algorithm reaches the upper limit. The expression of the function is

$$\mathrm{PI}(x) = P(f(x) \geq f(x^+)) = \Phi\left(\frac{\mu(x) - f(x^+)}{\sigma(x)}\right). \tag{10}$$

The idea of PI strategy is very simple. However, the drawbacks are obvious: It is a greedy algorithm which only considers exploration (sampling from the areas near the current optimal solution). Therefore, the sampling point is limited in a small range and easy to fall into the local optimal solution.

In order to solve this problem, a parameter ε is added into PI function. Only when the difference between the value of the next sampling point and the current optimal value is not less than ε, the new sampling point replaces the current optimal value. The extended PI function expression is defined as

$$\mathrm{PI}(x)=P\left(f(x)\geq f\left(x^{+}\right)+\varepsilon\right)=\Phi\left(\frac{\mu(x)-f(x^{+})-\varepsilon}{\sigma(x)}\right). \tag{11}$$

2) Expected improvement (EI)

Function EI calculates the expectation of the degree of improvement that a point can achieve when exploring the vicinity of the current optimum value. If the improvement of the function value is less than the expected value after the algorithm is executed, then the current optimal value point may be the local optimal solution, and the algorithm will find the optimum value point in other positions of the domain. Compared with PI function, EI function is not easy to fall into the local optimum solution.

The degree of improvement I is defined as the difference between the function value at the sampling point value and the current optimum value. If the function value at the sampling point value is less than the current optimum value, the improvement function is 0:

$$I(x)=\max\left\{0,f_{t+1}(x)-f\left(x^{+}\right)\right\}. \tag{12}$$

According to the EI function optimization strategy, we try to maximize EI with respect to the current optimum value $f(x^{+})$:

$$x=\arg\max E[I(x)]=\arg\max E(\max\left\{0,f_{t+1}(x)-f\left(x^{+}\right)\right\}. \tag{13}$$

When $f_{t+1}(x)-f(x^{+})\geq 0$, the distribution of $f_{t+1}(x)$ obeys the normal distribution with the mean $\mu(x)$ and the standard deviation $\sigma^{2}(x)$. Hence, the distribution of the random variable I is the normal distribution with the mean $\mu(x)-f(x^{+})$ and the standard deviation $\sigma^{2}(x)$. The probability density function of I is

$$f(I)=\frac{1}{\sqrt{2\pi}\sigma(x)}\exp\left(-\frac{(\mu(x)-f(x^{+})-I)^{2}}{2\sigma^{2}(x)}\right),\ I\geq 0. \tag{14}$$

EI is defined as

$$E(I)=\int_{\infty}^{\infty}If(I)\,\mathrm{d}I=\int_{I=0}^{I=\infty}I\,\frac{1}{\sqrt{2\pi}\sigma(x)}\exp\left(-\frac{(\mu(x)-f(x^{+})-I)^{2}}{2\sigma^{2}(x)}\right)\mathrm{d}I=\sigma(x)\left[Z\Phi(Z)+\varphi(Z)\right] \tag{15}$$

where

$$Z=\frac{\mu(x)-f(x^{+})}{\sigma(x)}. \tag{16}$$

Equation (15) denotes the expectation of the improvement I, which is the definition of EI function.

3) GP upper confidence bound (GP-UCB)

The function GP-UCB determines whether the next sampling point should make use of the current optimum value (corresponding to the high $\mu(x)$ zone) or should explore other low confidence zone (corresponding to the high $\sigma(x)$ zone). The parameter κ is used to make a trade-off between them. The definition of the function is defined as

$$\mathrm{UCB}(x)=\mu(x)+\kappa\sigma(x). \tag{17}$$

In order to determine the acquisition function, it is necessary to compare the performance of three acquisition functions with the same Gaussian process posterior. Fig. 2 shows the performance of PI, EI, and GP-UCB for optimizing the same function. *T* denotes the iteration steps; the dotted red line is the objective function value; the blue line represents the GP posterior mean; the red vertical line indicates the position of the current optimum value. It can be seen that after five iterations, both EI and GP-UCB functions find the global optimum value, while PI falls in the local optimum solution. According to the experimental results, we will use the EI function to optimize hyperparameters of the machine learning algorithms in terms of simplicity. Since on one side EI outperforms PI; on the other side EI does not need to tune other hyperparameters as GP-UCB does.

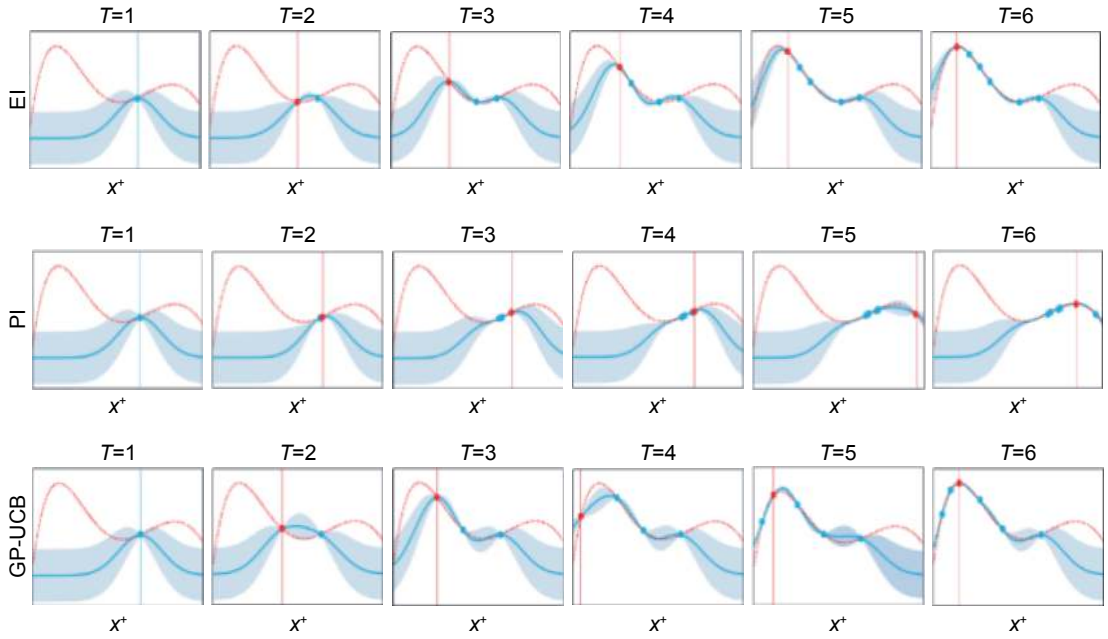


Fig. 2. Comparison of the optimization processes of three acquisition functions.

4. Hyperparameter Optimization for Machine Learning Models

According to Section 3, Bayesian optimization could find the optimal value through only a small number of samples. Compared with traditional optimization methods, it does not need the explicit expression of the function. Hence, Bayesian optimization is appropriate for tuning hyperparameters. In this section, Bayesian optimization algorithm is applied to optimize hyperparameters for three widely used machine learning models.

There are many machine learning models, e.g. discriminant analysis, support vector machine, decision tree, ensemble methods, etc. Reference [10] has evaluated the performance of 179 classifiers in the Machine Learning Repository (UCI) dataset[11], and the experiments showed that random forest algorithm is the optimal classifier among them. Therefore, firstly, Bayesian optimization is applied to tune hyperparameters for random forest algorithm; then, Bayesian optimization is used to tune hyperparameters of the artificial neural networks (including convolution neural network and recurrent neural network). Convolution neural network is widely used in deep learning architecture, which has excellent performance in solving the visual recognition problem. Recurrent neural network is a kind of neural network for processing sequential data. In the last part, we will apply Bayesian optimization algorithm to tune hyperparameters for deep forest which is a novel machine learning algorithm proposed in 2017. All the experiments will be carried out on the standard datasets.

4.1. Random Forest

Random forest[12] is a widely used ensemble algorithm for classification or regression tasks. The main principle of ensemble algorithms is based on that a group of weak learners can come together to form a strong learner. Random forest grows many classification trees with a standard machine learning technique called "decision tree". To classify a new sample, each tree outputs a classification and the final result is based on the vote of all trees. Random forest works as follows.

1) Suppose that the number of training sets is N. Sample N new samples at random with replacement from the original training set.

2) Suppose that the number of predictor variables is M. For each node in the decision tree, m predictor variables are selected out of all predictor variables, where $m<<M$. The predictor variable subset is produced by sampling at random. The best split on the predictor subset is used to split the node.

3) Repeat the above steps until n decision trees are built.

4) The final classification result of a new sample is determined by the vote of all decision trees.

Random forest is an improvement of the decision tree algorithm, and random forest model avoids over-fitting by randomness. The classification of single tree may be weak, but after quantity decision trees are built, the data classification result is determined by the mode of classification results of all trees. The advantages of random forest are obvious: It can provide high prediction accuracy and run fast on the large dataset. It can handle unbalanced and missing data.

It is noted that the performance of random forest is closely related to the strength of each tree and the inter-tree correlation. As m goes up, the strength of each tree can be improved, but the inter-tree correlation increases and the random forest error rate goes up. As m goes down, both inter-tree correlation and the strength of individual trees go down. So, the optimal value of m must be chosen carefully to make the trees as uncorrelated as possible. In addition, since the final classification result of algorithm is produced by all decision trees, the number of decision trees in the random forest will also affect the accuracy of the model. Usually, the performance of single tree in the forest is low, so if the number of decision trees is too small, the whole random forest model will have bad performance. Hence, the hyperparameters m and n must be chosen carefully.

Next, hyperparameters of random forest model are tuned by Bayesian optimization. We choose the number of decision trees in the random forest n and the size of the predictor variables subset m as hyperparameters. The default values of hyperparameters are n=10, $m = \sqrt{M}$ (M is the number of predictor variables). The ranges of value for hyperparameters are $n \in [1, 200]$ and $m \in [1, 20]$, respectively. The model prediction accuracy on the test set is chosen as the optimization function. It is averaged by 10 times 10-fold cross-validation. The Bayesian optimization process works as following. Firstly, select five sample points randomly in the hyperparameters space and calculate the prediction accuracy of the model. The five samples are used as the training set; next, obtain a new sample point by optimizing the acquisition function and calculate the acquisition function value at the new sample point; lastly, add the new sample point into the training set and update the posterior distribution of the function. Repeat the above steps until reaching the limit of iterations.

We select 7 datasets in UCI for the experiment. The UCI data set is a widely used dataset with high quality and quite reliable. The information about the datasets is shown in Table 2. Certain datasets are preprocessed by simple feature engineering. According to [2], grid search is reliable in low dimensional spaces (e.g., 1-dimension and 2-dimension). In this experiment, we will compare the performance of Bayesian optimization with grid search algorithm.

Experiment results are presented in Table 3. The experiment was implemented in an environment with a processor Intel® Core™ i5-4590 CPU, 3.30 GHz, memory size 8 G. The second column in Table 3 denotes the

default value of hyperparameters before optimization. The third column presents the accuracy of the model under default setting. The fifth and eighth columns show the mean prediction accuracy after three runs. The experiment results show that the accuracy of the model performance has a significant improvement by using hyperparameter optimization algorithms. Both Bayesian optimization and grid search perform almost equally well. However, Bayesian optimization runs faster than grid search.

Table 2: Information about selected UCI datasets

No.	Name	Number of samples	Number of predictor variables	Missing value?
1	Car evaluation	1228	7	No
2	Ks vs. Kp	5961	21	Yes
3	Keep balance	566	5	No
4	Vote dataset	1670	11	No
5	Nursery	12960	8	Yes
6	Mushroom	553	6	No
7	Titanic passenger	2224	12	Yes

Table 3: Comparison between Bayesian optimization and grid search

No.	Bayesian optimization					Grid search	
	Default values	Accuracy (%)	Optimized values	Accuracy (%)	Time (s)	Accuracy (%)	Time (s)
1	n=10, f=2	89.56	n=189, f=7	90.68	48	90.67	653
2	n=10, f=5	89.69	n=86, f=19	94.03	68	94.05	839
3	n=10, f=2	93.06	n=65, f=5	95.38	38	95.35	435
4	n=10, f=3	97.04	n=136, f=10	98.12	48	98.11	507
5	n=10, f=3	87.11	n=173, f=7	94.58	35	94.35	1035
6	n=10, f=2	93.65	n=97, f=4	97.33	39	97.13	488
7	n=10, f=3	63.75	n=88, f=5	72.51	55	72.81	565

4.2. Neural Networks

The convolutional neural network (CNN) is a particular type of deep, feedforward network for image recognition and classification algorithms[13]. It was developed in recent years and has achieved great success in the computer vision community. Compared with traditional neural networks, CNN has an advantage. It has less free parameters to learn for processing high-dimensional data, e.g. imagery data, which helps to accelerate the train speed and reduce the chance of overfitting. Hence, CNN has achieved great success in image recognition and semantic analysis domain[14]. CNN possesses the following distinguishing properties.

1) Local connections: Normally, local groups of data are often highly correlated, forming distinctive local motifs that are easily detected[15]. Each neuron is connected to local patches of the previous layer. Those local connections permit CNN to learn a spatially local pattern.

2) Shared weights: The local statistics of images and other signals are invariant to the location[15]. In other words, if a patch feature is useful to compute at some spatial positions, then it should also be useful to compute at other positions. The features learned by neural networks in a portion of the picture can also be used in other parts, for all of the locations on this image, the same learning feature is applicable.

3) Pooling: The role of pooling layer is to semantically merge similar features into one[15]. The most common operation is max pooling. The pooling layer serves to progressively reduce the spatial size of the representation. Hence, it reduces the number of parameters and the amount of computation in the network.

The recurrent neural network (RNN) is a class of artificial neural network for processing sequential data. Unlike feedforward networks, connections between neuron units form a directed cycle, which permits RNN to maintain information about the past elements of the sequence. RNN is good at language processing[16]. Using RNN to build

language model was proposed in [17]. Reference [18] proposed an improved version of RNN model, the long short-term memory (LSTM) network in 2012, which can store information for a very long time. At present, RNN has been widely applied in the fields of language model and text generation[19], speech recognition[20], and machine translation[21].

The performance of neural networks depends on a good setting for hyperparameters, such as the learning rate α, the momentum term β, the number of layers, the number of hidden units for the different layers, the learning rate decay, the mini-batch size, etc. Some of hyperparameters are more important than others. The learning rate is the most important hyperparameter to tune. Besides learning rate, a few other hyperparameters, such as the mini-batch size, the momentum term, and the hidden units, are also important. In the following experiments, we choose learning rate and mini-batch size as hyperparameters to tune. It is noted that the proposed method cannot be used to tune the number of layers, the number of nodes in each layer, and the size of filter. The reason is that the structure and connectivity of a network is a sequential decision problem. That means the early decision will influence the following decisions. Hence, Bayesian optimization is not appropriate for solving this kind of problem.

4.2.1. MNIST Dataset

This experiment uses data from MNIST database[22],[23]. This dataset has 28×28 pixels grey-scale images of handwritten digits, each belonging to one of ten classes. There are 70000 images in the dataset, with a training set of 60000 examples, and a test set of 10000 examples.

1) Hyperparameters tuning for CNN

In this experiment, CNN with two convolution layers and two pooling layers is used to process the data. The structure of the whole neural network is showed in Fig. 3. The size of convolution filter is 5×5, the stride is 1, zero-padding. The input and output of images have the same size. The pooling layers use a max-pooling filter with the size of 2×2. The last layer uses Softmax regression to compute the class score. Softmax regression is an extension of logistic regression to solve multiple classification problems[24].

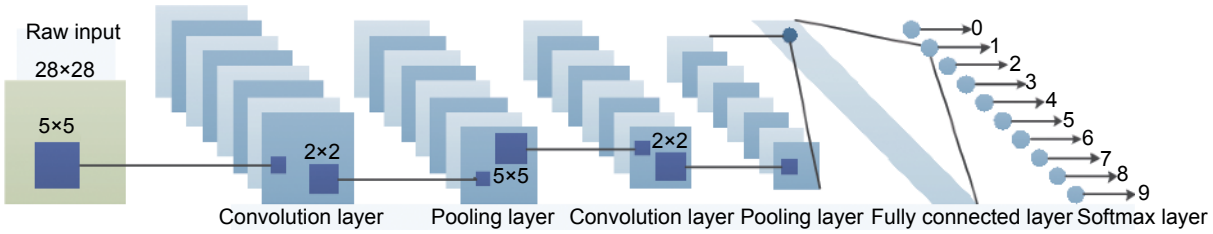


Fig. 3. Structure of CNN for processing MNIST dataset.

According to the above analysis of CNN, we finally select the learning rate of gradient descent algorithm α and the number of samples in each batch s as hyperparameters to be optimized. The ranges of the hyperparameters are $\alpha \in [0, 1]$ and $s \in [20, 2000]$. The prediction accuracy in the test set is set as the optimization function. Before executing the algorithm, five sample points are randomly selected in the space of hyperparameters and their model prediction accuracy is calculated on the test set, then the point with the highest accuracy is selected as the initial values of the model parameters.

Fig. 4 presents the final result of hyperparameters optimization. It records the average prediction accuracy of CNN after each iteration of Bayesian optimization. The experiment was implemented three times. The horizontal axis represents the iterations of Bayesian optimization algorithm; the vertical axis represents the prediction accuracy on the test set. The model accuracy is 98.83% before tuning hyperparameters and it reaches 99.14% after 50 iterations.

2) Hyperparameters tuning for RNN

In this experiment, RNN is constructed with one hidden layer. The structure of the whole neural network is shown as Fig. 5. The number of nodes in the hidden layer is 128, the number of nodes in the input layer is 28, and that in the output layer is 10. The time step is 28.

The learning rate of gradient descent algorithm α and the number of samples in each batch s are chosen to be optimized. The ranges of the values are $\alpha \in [0, 1]$ and $s \in [20, 2000]$. The prediction accuracy after hyperparameters optimization is shown in Fig. 6. The model accuracy is 91.20% before tuning hyperparameters and the model accuracy achieves prediction accuracy of 97.13% after 50 iterations.

4.2.2. *CIFAR-10 Dataset*

CIFAR-10 is an object recognition dataset created by Krizhevsky and Hinton[25]. The dataset consists of 60000 32×32 color images in 10 classes with 6000 images per class. There are 50000 training images and 10000 test images.

In this experiment, the model proposed by Krizhevsky in [26] is used. The model is composed by two convolution layers, two nonlinear layers, and a fully-connected layer. Fig. 7 presents the structure of the neural network. The size of convolution filter is 5×5 and the stride is 1. The pooling layers use the max-pooling with the size of 4×4.

The learning rate of gradient descent algorithm and the decay time of learning rate are chosen as the parameters to be optimized. The ranges of hyperparameters are $\alpha \in [0, 1]$ and $d \in [100, 1000]$. The goal of optimization is to maximize the prediction accuracy of the neural network in the test set.

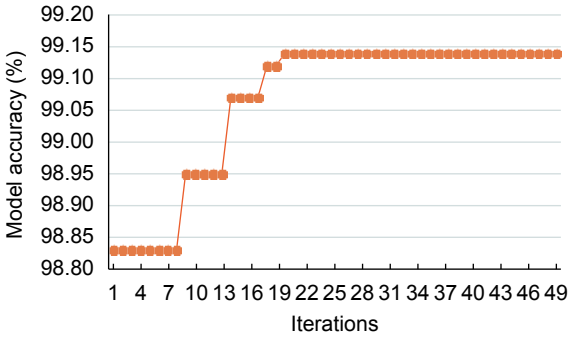


Fig. 4. Prediction accuracy of CNN after hyperparameters optimization on MNIST.

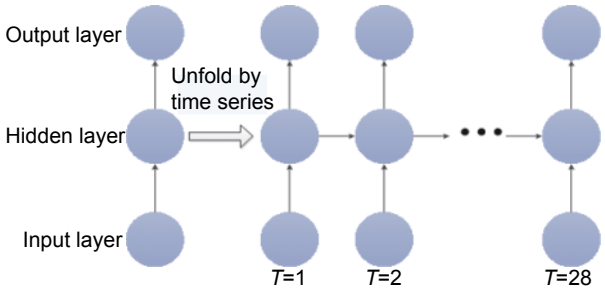


Fig. 5. Structure of RNN for processing MNIST database.

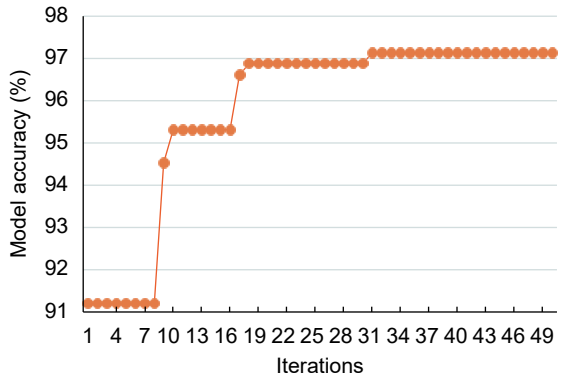


Fig. 6. Prediction accuracy of RNN after hyperparameters optimization on MNIST.

The final result of hyperparameters optimization is shown in Fig. 8. The model accuracy is 86.58% before optimizing hyperparameters, and the prediction accuracy achieves 88.41% after 50 iterations.

4.3. Multi-Grained Cascade Forest

Multi-grained cascade forest (gcForest) is a novel decision tree ensemble method integrated proposed by [27]. Compared with the deep neural networks, gcForest can achieve higher accuracy, has less hyperparameters to tune, and needs a less training time.

gcForest is a cascade structure consisted of multiple levels. Each level is an ensemble of decision tree forests, which includes random forest and complete random forest. For each instance, each forest outputs an estimate of the class distribution. The class vector concatenates with the original feature vector to be inputted to the next level.

In order to prevent overfitting, the model will be tested on a test set after training every level. The training procedure will be terminated if there is no significant performance gain.

Because the basic structure of gcForest is an ensemble of random forests, according to the analysis of the random forest, we selected the number of decision trees in a random forest n, the size of feature subset f, and the number of forests in each level t, as hyperparameters to tune. The ranges of the values are $n \in [100, 500]$, $f \in [1, 20]$, and $t \in [2, 20]$. The model predictive accuracy in the test set is selected as the optimization function.

The performance of gcForest is evaluated on two UCI datasets with a relatively small number of features: Dataset LETTER with 16 features and 16000/4000 training/test examples and dataset ADULT with 14 features and 32561/16281 training/test examples. The experiment results are summarized in Table 4. Before tuning hyperparameters, the prediction accuracy on LETTER and ADULT is 96.15% and 85.68%, respectively. After tuning hyperparameters by Bayesian optimization, the prediction accuracy is improved, which is 97.68% on LETTER and 87.05% on ADULT, compared with the experiment results in [27] where gcForest achieves 97.40% and 86.40% on LETTER and ADULT, respectively. Bayesian optimization definitely improves the prediction accuracy of gcForest.

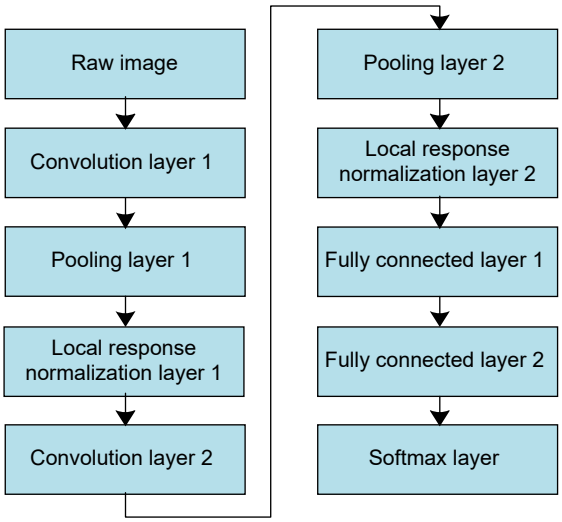


Fig. 7. Structure of the improved convolutional neural network proposed in [26].

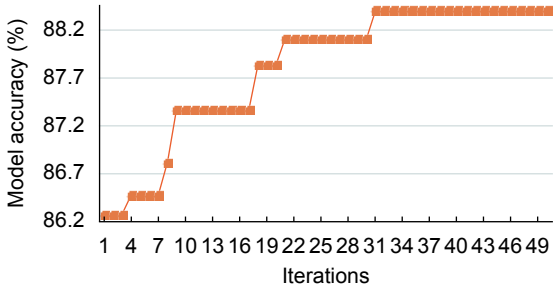


Fig. 8. Prediction accuracy of CNN model proposed in [26] after hyperparameters optimization on CIFAR-10.

Table 4: Bayesian optimization for gcForest

Dataset	Before optimization	After optimization
LETTER	Prediction accuracy: 96.15% Default hyperparameters: n=100, f=4, t=2	Prediction accuracy: 97.68% Hyperparameters optimized: n=357, f=10, t=8
ADULT	Predictive accuracy: 85.68% Default hyperparameters: n=100, f=4, t=2	Predictive accuracy: 87.05% Hyperparameters optimized: n=402, f=9, t=12

5. Conclusion

This paper presented a hyperparameter tuning algorithm for machine learning models based on Bayesian optimization. Bayesian optimization combined a prior distribution of a function with sample information (evidence) to obtain posterior of the function; then the posterior information was used to find where the function was maximized according to a criterion. Several experiments were conducted on standard datasets. Experimental results show that the Bayesian optimization algorithm based on Gaussian process can achieve great accuracy in a few samples. The runtime is also significantly reduced compared with manual search.

References

[1] J. Aarshay. (2018). Complete guide to parameter tuning in gradient boosting (GBM) in python. [Online]. Available: https://www.cnblogs.com/peizhe123/p/8124943.html

[2] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281-305, 2012.

[3] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. of the 24th Intl. Conf. on Neural Information Processing Systems*, 2011, pp. 2546-2554.

[4] B. Betrò, "Bayesian methods in global optimization," *Journal of Global Optimization*, vol. 1, no. 1, pp. 1-14, 1991.

[5] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces," *Journal of Global Optimization*, vol. 21, no. 4, pp. 345-383, 2001.

[6] E. Brochu, V. M. Cora, and N. D. Freitas. (2010). A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. [Online]. Available: https://arxiv.org/pdf/1012.2599.pdf

[7] J. Joyce, "Bayes' theorem," *Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed. Stanford: Stanford University, 2003.

[8] P. J. Lamberson, "Exploration and exploitation," *Differential Evolution: In Search of Solutions*, V. Feoktistov, Ed. Boston: Springer, 2006, pp. 69-81.

[9] C. E. Rasmussen and C. K. I. Williams, "Gaussian processes for machine learning," *Intl. Journal of Neural Systems*, vol. 103, no. 14, pp. 429-429, 2006.

[10] M. Fernandez-Delgado, E. Cernadas, and S. Barro, "Do we need hundreds of classifiers to solve real world classification problems?" The Journal of Machine Learning Research, vol. 15, no. 1, pp. 3133-3181, 2014.

[11] C. Newman and C. Merz. (1998). UCI repository of machine learning databases. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

[12] L. Breiman, "Random forest," *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. of the 25th Intl. Conf. on Neural Information Processing Systems*, 2012, pp. 1097-1105.

[14] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," in *Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014, p. 1.

[15] Y. LeCun, Y. Bengio, and H. Geoffrey, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436-444, 2015.

[16] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137-1155, 2003, DOI: 10.1007/3-540-33486-6-6

[17] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. of Interspeech*, 2010, pp. 1045-1048.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997.

[19] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proc. of the 28th Intl. Conf. on Machine Learning*, 2011, pp. 1017-1024.

[20] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proc. of the 31th Intl. Conf. on Machine Learning*, 2014, pp. 1764-1772.

[21] S.-J. Liu, N. Yang, M. Li, and M. Zhou, "A recursive recurrent neural network for statistical machine translation," in *Proc. of the 52th Annual Meeting of the Association for Computational Linguistics*, 2014, pp. 1491-1500.

[22] H.-Y. Wang and S. Bengio. (2002). The MNIST database of handwritten upper-case letters. [Online]. Available: https://infoscience.epfl.ch/record/82790/files/com02-04.pdf

[23] Y. Lecun and C. Cortes. (2010). The MNIST database of handwritten digits. [Online]. Available: https://www.lri.fr/~marc/Master2/MNIST_doc.pdf

[24] T. G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," *Journal of Artificial Intelligence Research*, vol. 2, no. 1, pp. 263-286, 1994.

[25] A. Krizhevsky and G. Hinton. (2009). Learning multiple layers of features from tiny images. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.222.9220

[26] A. Krizhevsky. (2010). Convolutional deep belief networks on cifar-10. [Online]. Available: https://www.cs.toronto.edu/~kriz/conv-cifar10-aug2010.pdf

[27] Z.-H. Zhou and J. Feng. (2017). Deep forest: Towards an alternative to deep neural networks. [Online]. Available: https://arxiv.org/abs/1702.08835v2

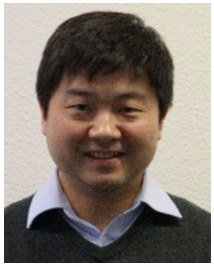
Jia Wu was born in Sichuan Province, China in 1980. She received the M.S. degree in computer science from University of Electronic Science and Technology of China, Chengdu, China in 2006, and the Ph.D. degree in automation from Université de Technologie de Belfort-Montbéliard, Belfort, France in 2011. She is currently an associate professor with University of Electronic Science and Technology of China. Her research interests include deep reinforcement learning, data mining, and intelligent transportation systems.

Xiu-Yun Chen was born in Jiangxi Province, China in 1993. He received the B.S. degree from Jiangxi University of Science and Technology, Jiangxi, China in 2012. He is currently pursuing the M.S. degree with University of Electronic Science and Technology of China. His research interests include deep learning and reinforcement learning.

Hao Zhang was born in 1993. He received the M.S. degree in computer science from University of Electronic Science and Technology of China in 2018. His research interests include data mining and machine learning.

Li-Dong Xiong was born in Sichuan Province, China in 1989. He received the B.S. degree from University of Electronic Science and Technology of China in 2008. He is currently pursuing the M.S. degree with University of Electronic Science and Technology of China. His research interests include reinforcement learning and big data processing.

Hang Lei was born in 1960. He is currently a professor and Ph.D. supervisor with University of Electronic Science and Technology of China. His research interests include embedded system design (hardware and software) and embedded software reliability.



Si-Hao Deng received the Ph.D. degree from Université de Technologie de Belfort-Montbéliard in 2007. He is currently an associate professor with Université de Technologie de Belfort-Montbéliard. His research interests include automation & robotics and artificial intelligence.