Computer Assisted Methods in Engineering and Science
32(1): 3–24, 2025, https://doi.org/10.24423/cames.2025.1730

Comparison of Particle Swarm Optimization Algorithms in Hyperparameter Optimization Problem of Multi Layered Perceptron

Kenta SHIOMI, Tetsuya SATO, Eisuke KITA*

Graduate School of Infomatics, Nagoya University, Nagoya 464-8601, Japan
** Corresponding Author e-mail: kita@i.nagoya-u.ac.jp*

This paper describes the application of particle swarm optimization (PSO) for the hyperparameter optimization problem of multi-layered perceptron (MLP) model. Several PSO algorithms are presented by many researchers; basic PSO, PSO with inertia weight (PSO-w), PSO with constriction factor (PSO-cf), local PSO-w, local PSO-cf, union of local and global PSOs (UPSO), PSO with second global best particle (SG-PSO), and PSO with second local best particle (SP-PSO). The wine dataset is taken as a numerical example and hyperparameters of MLP the model are determined by the above-mentioned PSO algorithms. The sets of hyperparameters determined by these PSO algorithms are compared with the results of the traditional algorithms for hyperparameter optimization such as random search, tree-structured Parzen estimator (TPE), and covariance matrix adaptation evolution strategy (CMA-ES).

Numerical results indicate that PSO-cf is the best-performing and local PSO-w is the second best among the PSO algorithms. The sets of hyperparameters determined by the PSO algorithms were relatively similar. An important finding from the numerical results is that PSO algorithms could find better hyperparameters than random search, TPE, and CMA-ES. This demonstrates that PSO is suitable for the hyperparameter optimization problem in MLP models.

Keywords: multi layered perceptron (MLP), hyperparameter optimization, particle swarm optimization (PSO), wine dataset.

1. Introduction

Neural network models have evolved with the proposal of deep learning, and now are widely applied to various problems [1, 2]. In order to improve the prediction accuracy of neural network models, it is necessary to appropriately set hyperparameters [3–5]. Therefore, in recent years, research on hyperparameter optimization methods has been actively conducted.

Several methods are applied for the hyperparameter optimization problems of neural network models; grid search, random search, Bayesian optimization, and evolutionary computation including genetic algorithm (GA) and particle swarm optimization (PSO) [6–9].

This study explores the application of PSO to hyperparameter optimization problems. The application of PSO to hyperparameter optimization in neural network models has been presented by several researchers [10–13]. They apply the PSO algorithm to the hyperparameter optimization of convolutional neural network (CNN) and deep neural network (DNN) models. However, various PSO algorithms have been proposed by several researchers over time. The PSO algorithms used in this research include basic PSO [14], which is the original algorithm, PSO with inertia weight (PSO-w) [15], PSO with constriction factor (PSO-cf) [16], local-PSO-w, local-PSO-cf [17], union of global and local PSOs (UPSO) [18], PSO with second global best particle (SG-PSO), and PSO with second personal best particle (SP-PSO) [19]. A discriminant problem from the wine dataset using multi layered perceptron (MLP) model is taken as an analysis example and, then, the hyperparameter optimization problem of the MLP model is solved by the aforementioned PSO algorithms. The results are compared with those obtained by random search, tree-structured Parzen estimator (TPE) [20] and covariance matrix adaptation evolution strategy (CMA-ES) [21].

The structure of this paper is as follows: the MPL model is explained in Sec. 2. PSO algorithms, TPE and CMA-ES [21] are introduced in Sec. 3. The solution of the hyperparameter optimization for the MLP model using PSOs is defined in Sec. 4. Section 5 discusses the experimental results. Section 6 provides the conclusion and outlines directions for future work.

2. Multi-layered perceptron (MLP)

2.1. Network structure

An MLP is a network with multiple layers, created by adding intermediate layers to a single-layer perceptron (Fig. 1).

The relationship between the input and output in the generalized MLP hidden layer is given as follows:

$$X_{i+1} = \phi(W_i^T X_i), \tag{1}$$

where i is the layer number ($i = 1, ..., N$), X_{i+1} is the output vector of the hidden layer, X_i is the input vector of the hidden layer, W_i^T represents the transposed weight vector of the hidden layer, and ϕ denotes the activation function such as a sigmoid function, rectified linear unit (ReLU) function, LeakyReLU function, and exponential linear unit (ELU) function.

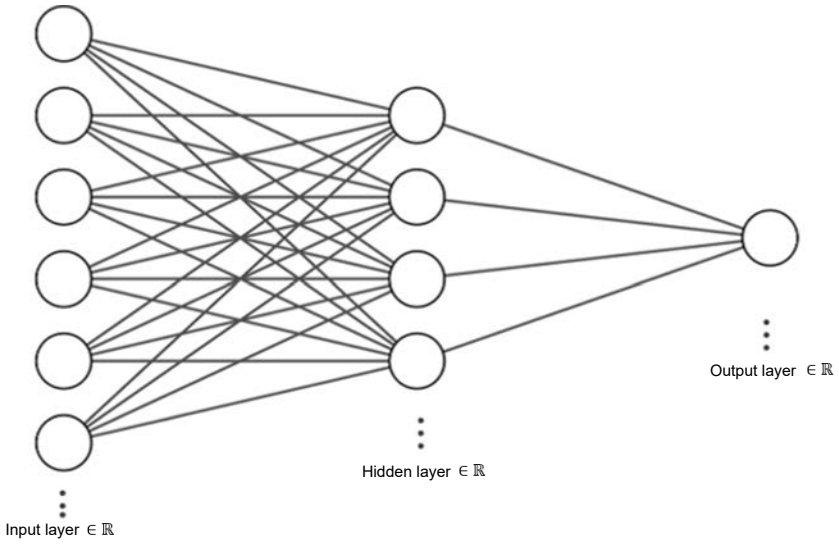

FIG. 1. Network structure of an MLP.

2.2. MLP hyperparameters

Multi-layered perceptron hyperparameters can be broadly classified into hyperparameters related to the hidden layer, activation function, batch size, and regularization [7–9, 22].

2.2.1. Hidden layer. There are two main types of hyperparameters related to the hidden layer. These are its depth and width. The depth of the hidden layer refers to the number of hidden layers, and the width refers the number of nodes in each hidden layer. The larger their values are, the better the expressive power of MLP will be. But, on the other hand, overfitting to training data may occur. Therefore, the depth and width of the intermediate layer should not be arbitrarily large. Instead, they must be adjusted through trial and error to maximize the generalization performance of the model, depending on the problem at hand.

2.2.2. Activation function. Hyperparameters in activation functions mainly correspond to the types of activation functions. There are many types, but the most representative ones include the sigmoid function (Eq. (2)), softmax function (Eq. (3)), and ReLU function (Eq. (4)).

The sigmoid and softmax functions are nonlinear functions, whereas ReLU function, which does not cause the vanishing gradient problem, is increasingly used in the $x > 0$ region.

The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{(1 + e^{-a})}, \tag{2}$$

where the threshold a is a hyperparameter of the sigmoid function, which is generally set to 1. A feature of the sigmoid function is that it can convert the output value into a range of 0 to 1, making it suitable for converting the final output value into a probability value in binary classification. Another problem with the sigmoid function is that when it is introduced as an activation function in the hidden layer, a vanishing gradient problem occurs, making optimization in the hidden layer difficult. In contrast, the softmax function is an activation function that converts output values into probability values in classification of two or more classes.

The softmax function is defined as:

$$\mathrm{softmax}(x_k) = \frac{e^{x_k}}{\sum_j e^{x_j}}, \tag{3}$$

where k is the serial number of input values ($k = 1, ..., m$), e^{x_k} is the output when the k-th input vector is processed through the exponential function, $\sum_j e^{x_j}$ is the sum of outputs when all input vectors are passed through the exponential function as input. The softmax function has roughly the same characteristics as the sigmoid function, but its major feature is that it can convert multiple input values into probability values.

The ReLU function is defined as:

$$\mathrm{ReLU}(x) = \begin{cases} x \ (x \geq 0), \\ 0 \ (x < 0). \end{cases} \tag{4}$$

The ReLU function is highly effective as an activation function for intermediate layer. However, one limitation is that it cannot be learned using the gradient when the activation function becomes 0. Therefore, derivative models of the ReLU function such as the leaky ReLU function, randomized ReLU (RReLU) function, and parametric ReLU (PReLU) function are used.

2.2.3. Optimizer for weighting coefficients and thresholds. Neural network models use optimization algorithms or optimizers to adjust parameters such as weighting coefficients and thresholds. The parameter that determines the update frequency of the parameters is called the learning rate. There are various types of optimization algorithms, including stochastic gradient algorithm (SGD), momentum, adaptive gradient propagation (AdaGrad), root mean squared propagation (RMSProp), and adaptive momentum estimation (Adam).

2.2.4. Batch size. In parameter learning for neural network models, the training data is divided into training data subsets called mini-batches, and batch normalization is applied to them.

The amount of data required to update the parameters in one iteration is called the batch size. When the batch size is equal to the total number of training data, it is called batch learning. When the batch size is smaller than the number of training data, it is called mini-batch learning. Batch learning uses all the training data at once, so it has the disadvantage of increasing computational costs. It is common to use mini-batch learning. Commonly used batch sizes are 16, 32, 64, 128, 256, 512, 1024, and larger.

2.2.5. Normalization. Normalization is a data transformation technique that transforms data into a distribution with a mean of 0 and a variance of 1. Batch normalization is normalization performed in batches in each dimension. By introducing batch normalization, it is expected that the learning speed will be improved.

There are various types of normalization methods, including batch normalization, layer normalization, instance normalization, and group normalization.

2.2.6. Regularization and dropout. Measures against model overfitting include a norm penalty (a regularization term) and dropout. A norm is an index for measuring the distance of vectors, including $L0$ norm, $L1$ norm, $L2$ norm, $L\infty$ norm, and so on,

$$\|x\|_p = (|x_1| + |x_2|^p + \cdots + |x_D|^p)^{1/p}, \qquad p = 0, 1, 2, ..., \infty. \tag{5}$$

Norm penalty is a method of limiting parameter values based on norms so that they do not take extreme values,

$$L_{\text{norm}}(y(x), t) = L(y(x), t) + \lambda \|x\|_p, \tag{6}$$

where $L(y(x), t)$ is the error function, λ is the weight of the norm penalty, and $L_{\text{norm}}(y(x), t)$ is the error function with the norm penalty. Typical norm penalty methods include lasso regression ($L1$ norm) and ridge regression ($L2$ norm). Norm penalties are not required in model learning and are determined by the model designer. When using a norm penalty, it is necessary to appropriately adjust both the type of norm penalty and the hyperparameter λ of the norm penalty.

Dropout, on the other hand, is a method that prevents model overfitting by randomly eliminating a certain percentage of nodes. By using dropout, one can reduce biased learning that depends on specific nodes. This can prevent the model from learning overly detailed features of the training data.

2.2.7. Number of epochs. The number of epochs, which refers to the number of training times the model is trained, directly affects the generalization performance of the model. The more times a model learns based on the entire training data, the greater the risk of overfitting, where the model overfits the training data. If the number of epochs is too small, the model will be in an unlearning state, where it has not sufficiently learned the data, and in neither case it cannot be said that the model learns properly. Therefore, model designers need to adjust the number of epochs to ensure the model can learn appropriately.

3. Optimization algorithms of hyperparameters

3.1. Particle swarm optimization

Particle swarm optimization (PSO), introduced by Kennedy and Eberhart in 1995 [14], is an optimization method inspired by the behavior of birds and fish to efficiently search for food in groups. Individuals, such as birds and fish, that behave in groups, are called particles, and groups of particles from what is called particle swarms.

Particles have a position that represents the solution to the optimization problem and a velocity that updates the position. In the basic PSO, at the beginning of the PSO algorithm, the position and velocity of a particle are updated by the following equations:

$$x_i(t+1) = x_i(t) + v_i(t+1), \tag{7}$$

$$v_i(t+1) = v_i(t) + c_1 r_1 (x_i^p(t) - x_i(t)) + c_2 r_2 (x^g(t) - x_i(t)), \tag{8}$$

where i is the particle number ($i = i, ..., N$), N is the number of particles, and t is the time step. $x^p(t)$ and $x^g(t)$ refer to the personal best and global best particles, respectively. The personal best and global best particles denote the solution that each particle has found so ever and the solution that all particles in the swarm have found, respectively. c_1 and c_2 are the weight parameters related to the personal best and global best particles, respectively. r_1 and r_2 are random numbers belonging to different random number sequences in the range $[0, 1]$.

The basic PSO algorithm is summarized as follows:

1. Definition of the number of generations:
   - Define the maximum number of generations $t_{\max}$.
   - Initialize the number of generations t as $t = 0$.
2. Initialize particle swarm:
   - Define the number of particles N.

- Initialize the position $x_i(0)$, of each particle, with a random number within the range that satisfies the constraints of each design variable.
- Initialize the velocity $v_i(0)$ of each particle to 0.

3. Fitness evaluation:
   - Evaluate the fitness function from the particle position information.
4. Terminal judgment:
   - If $t = t_{\max}$, terminate the process. Otherwise, proceed to the next step.
5. Update of personal best particle:
   - If $t = 0$, set the particle evaluated in step 3 as the personal best particle. If $t > 0$, compare the fitness value of each particle calculated in step 3 with the personal best particle from the generation and update the particle with the better fitness value as the personal best particle, as follows:

$$S^p(t) = S_j^p(t) = \begin{cases} x_i(t), & t = 0, \\ \{x_i^p(t-1), x_i(t)\}, & t > 0, \end{cases} \tag{9}$$

$$x_i^p(t) \leftarrow \arg \min_{S_j^p(t) \in S^p(t)} f(S_j^p(t)), \tag{10}$$

   where i, N and t are the particle number, the number of particles, the number of generations, respectively. $x^p(t)$ and $f(x)$ denote the personal best particle and the fitness function, respectively.
6. Update of global best particle:
   - If $t = 0$, set the particle with the best fitness value from the personal best particles evaluated in step 5 as the global best. If $t > 0$, compare the personal best particles updated in step 5 with the global best particle from the previous and set the particle with the better evaluation value as the global best, as follows:

$$S^p(t) = S_j^p(t) = \begin{cases} x_i(t), & t = 0, \\ \{x_i^p(t-1), x_i(t)\}, & t > 0, \end{cases} \tag{11}$$

$$x^g(t) \leftarrow \arg \min_{S_j^g \in S^g} f(S_j^g), \tag{12}$$

   where i, N and t are the particle number, the number of particles, the number of generations, respectively. $x^p(t)$, $x^g(t)$ and $f(x)$ denote the personal best particle the global best particle, and the fitness function, respectively.

7. Update the position and velocity of each particle by Eqs. (7) and (8), respectively.
8. Update the number of generations:
   - Update the current number of generations t as $t \leftarrow t+1$ and to step 3.

3.1.1. Basic PSO. As described in the previus subsection, the basic PSO uses the velocity update rule given by Eq. (8) [14].

3.1.2. PSO with inertia weight. PSO with inertia weight (PSO-w) is an algorithm that introduces an inertia term w into the velocity update equation of basic PSO [15]. The inertia term w suppresses the velocity of particles as the generations progress and it acts to balance the search range and search speed. Increasing the value of the inertia term w improves the search performance across the entire particle swarm solution space. It, however, reduces the convergence speed. On the other hand, reducing the value of the inertia term w limits the search performance in the entire solution space, but improves the convergence speed. The update equations for velocity is given as follows:

$$v_i(t+1) = wv_i(t) + c_1 r_1 (x_i^p(t) - x_i(t)) + c_2 r_2 (x^g(t) - x_i(t)). \tag{13}$$

The inertia term w generally takes a large value at the initial stage of the search and gradually decreases [15]. The purpose of this is to focus the search performance for the entire solution space in the early stages of the search, and to improve the search performance around the optimal solution in the final stages of the search. The formula for calculating the value of the inertia term w according to the value of the number of generations is given:

$$w = w_{\max} - (w_{\max} - w_{\min}) \times \frac{t}{t_{\max}}, \tag{14}$$

where $w_{\max}$ and $w_{\min}$ are the maximum and minimum values of the inertia term w. t and $t_{\max}$ are the current number of generations and the upper limit of the number of generations, respectively.

3.1.3. PSO with constriction factor (PSO-cf). This algorithm was proposed by Clerc and Kennedy and aims to stabilize the behavior of particle groups by introducing a convergence coefficient K [16]. It has been suggested that it is effective to use the convergence coefficient K for ensuring the convergence of the particle group in the PSO algorithm. The update formula for velocity is shown:

$$v_i(t+1) = K[v_i(t) + c_1 r_1 (x_i^p(t) - x_i(t)) + c_2 r_2 (x^g(t) - x_i(t))]. \tag{15}$$

The convergence coefficient K is defined by the following equation:

$$K = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|}. \tag{16}$$

The value of the convergence coefficient K is determined by the value of φ, and the value of φ must be greater than 4.0. However, as the value of φ increases, the value of the convergence coefficient K decreases, and the diversity of particles decreases. Therefore, the typical value of φ is $c_1 = c_2 = 2.05$, and it is generally defined as $\varphi = 4.1$ [16].

3.1.4. Local-PSO-w/local-PSO-cf. The PSO algorithm described above has the property that its local search ability is poor at the beginning of the search, and its overall search ability of the solution space is poor at the end of the search. Therefore, local PSO-w and local PSO-cf are proposed with the aim of balancing global search and local search capabilities in the solution space. These algorithms employ a new concept called the local best particle, which replaces the global best particle when updating the position and velocity of the proposed particle [17]. The local best particle refers to the particle with the best solution among the neighboring particles of the particle being updated. Kennedy and Mendest suggest that reducing the number of neighboring particles leads to better search for solutions in complex optimization problems, while increasing the number of neighboring particles allows for faster search in simpler optimization problems. The update formula of the velocity is given as follows [17]:

- local-PSO-w

$$v_i(t+1) = wv_i(t) + c_1r_1(x_i^p(t) - x_i(t)) + c_2r_2(x_i^l(t) - x_i(t)); \tag{17}$$

- local-PSO-cf

$$v_i(t+1) = K[v_i(t) + c_1r_1(x_i^p(t) - x_i(t)) + c_2r_2(x_i^l(t) - x_i(t))]. \tag{18}$$

In these equations, x_i^l refers to the position of the local best particle, which is the particle with the best solution among the neighboring particles. In this study, the number of the neighboring particles is $N_P = 5$.

3.1.5. Union of global and local PSOs (UPSO). This algorithm probabilistically selects two velocity update patterns to update the particle velocity; one is based on the global best particle and the other is based on the local best particle [18]. The update formula for velocity is given as:

$$v_i(t+1) = uG_i(t+1) + (1-u)L_i(t+1), \tag{19}$$

where $G_i(t+1)$ and $L_i(t+1)$ are the velocities calculated from the global best and local best particles, respectively. The respective velocity is determined by the parameter u. In UPSO based on the PSO-w algorithm, $G_i(t+1)$ and $L_i(t+1)$ are defined as follows:

$$G_i(t+1) = wG_i(t) + c_1r_1(x_i^p(t) - x_i(t)) + c_2r_2(x^g(t) - x_i(t)), \tag{20}$$

$$L_i(t+1) = wL_i(t) + c_1r_1(x_i^p(t) - x_i(t)) + c_2r_2(x_i^l(t) - x_i(t)), \tag{21}$$

where $x^g(t)$ and $x_i^l(t)$ represent the global and local best particle positions, respectively.

3.1.6. PSO with second global best particle (SG-PSO). In order to solve the problem of convergence to a local solution in the conventional PSO method, the value of the second global best, which has the second best solution in the particle group, is used to update particle velocity information [19]. The update formula for velocity is given as:

$$v_i(t+1) = \begin{cases} wv_i(t) + c_1r_1(x_i^p(t) - x_i(t)) \\ \qquad + c_2r_2(x^g(t) - x_i(t)) + c_3r_3(x^{g2}(t) - x_i(t)), & r \le 0.5, \\ wv_i(t) + c_1r_1(x_i^p(t) - x_i(t)) + c_2r_2(x^g(t) - x_i(t)), & r > 0.5, \end{cases} \tag{22}$$

where $x^g(t)$ and $x^{g2}(t)$ denote the global best and second global best particle positions, respectively. c_3 is the weighting coefficient of the velocity update formula considering the second global best, and r and r_3 are random numbers belonging to different random number sequences in the interval [0,1].

3.1.7. PSO with second personal best particle (SP-PSO). In this method, in order to solve the problem of convergence to a local solution in the conventional PSO method, the value of the second personal best, which is the second best solution found by each particle, is used to update the particle velocity information [19]. In SP-PSO, the update formula for position and velocity is given as follows:

$$v_i(t+1) = \begin{cases} wv_i(t) + c_1r_1(x_i^p(t) - x_i(t)) \\ \qquad + c_2r_2(x^g(t) - x_i(t)) + c_4r_4(x^{p2}(t) - x_i(t)), & r \le 0.5, \\ wv_i(t) + c_1r_1(x_i^p(t) - x_i(t)) + c_2r_2(x^g(t) - x_i(t)), & r > 0.5, \end{cases} \tag{23}$$

where $x_i^p(t)$ and $x_i^{p2}(t)$ denote the personal best and the second personal best particles, respectively. c_4 is the weighting coefficient in the velocity update formula considering the second personal best particle, and r and r_4 are random numbers belonging to different random number sequences in the range [0,1].

The second global best in SG-PSO is a single piece of information common to a group of particles, but the second personal best in SP-PSO consists of multiple pieces of information, each defined for an individual particle. Therefore, it is suggested that the SP-PSO particle group can search a wider solution range than the SG-PSO particle group [19].

3.2. Random search

Random search is an optimization method that randomly searches for combinations of searchable solutions using a random number generator. Characteristics of random search include that all evaluations can be performed asynchronously and that the search space can be of a wide variety of variable types, including continuous, discrete, and categorical.

3.3. Tree-structured parzen estimator (TPE)

TPE is an algorithm classified as Bayesian optimization. This method consists of two components: a probabilistic surrogate model and an acquisition function [20]. The distribution of objective functions and hyperparameters is expressed from observed data using a surrogate model, and promising hyperparameters are efficiently searched for using an acquisition function. TPE employs a surrogate model based on kernel density estimation to perform Bayesian optimization.

3.4. Covariance matrix adaptation evolution strategy (CMA-ES)

Since CMA-ES is a type of evolutionary computation, the basic algorithm involves generating individuals and evaluating the fitness of each individual [21]. If the objective is also to optimize hyperparameters, each individual will maintain a combination of hyperparameters, and the fitness of the individual will be calculated based on the defined objective function. In CMA-ES, individuals are generated from a normal distribution, meaning the information they hold is only of continuous values. Therefore, when dealing with discrete values or categorical values, it is necessary to discretize or categorize the continuous values.

4. Hyperparameter optimization using PSO

4.1. Dataset

The wine dataset, sourced from the UCI Machine Learning Repository, is used for the experiments in this study. The wine dataset has 178 data items. The explanatory variables are 13 wine characteristics, which are based on the results of chemical analysis of three different wine varieties grown in the same

region of Italy. The 13 explanatory variables are alcohol content, malic acid, ash content, ash alkalinity, magnesium, total phenolic content, flavonoids, non-flavonoid phenols, proanthocyanins, color intensity, hue, absorbance ratio of diluted wine solution, and proline. Of these, only magnesium and proline take integer values, and all other explanatory variables take continuous values. The objective variables are the three types of wine, which form the basis for the explanatory variables. The objective variables are classified into classes 1, 2, and 3, and the number of data items for three classes is 59, 71, and 48, respectively. Table 1 shows the data for the first five items of the wine dataset.

TABLE 1. Explanatory variables of the wine dataset.

Name	Type	Value
Alcohol content	Continuous	14.23, 13.20, 13.16, 14.37, 13.24
Malic acid	Continuous	1.71, 1.78, 2.36, 1.95, 2.59
Ash content	Continuous	2.43, 2.14, 2.67, 2.50, 2.87
Ash alkalinity	Continuous	15.60, 11.20, 18.60, 16.80, 21.00
Magnesium content	Discrete	127, 100, 101, 113, 128
Phenol content	Continuous	2.80, 2.65, 2.80, 3.85, 2.80
Flavonoids	Continuous	3.06, 2.76, 3.24, 3.49, 2.69
Non-flavonoid phenols	Continuous	0.28, 0.26, 0.30, 0.24, 0.39
Proanthocyanidin	Continuous	2.29, 1.28, 2.81, 2.18, 1.82
Color intensity	Continuous	5.64, 4.38, 5.68, 7.80, 4.32
Hue	Continuous	1.04, 1.05, 1.03, 0.86, 1.04
OD280/OD315 of diluted wine solution	Continuous	3.92, 3.40, 3.17, 3.45, 2.93
Proline	Discrete	1065, 1050, 1185, 1480, 735

4.2. Network architecture and hyperparameters

The architecture of the MLP model is shown in Fig. 2. This architecture consists of 2 to 10 fully connected layers. Components highlighted in blue in

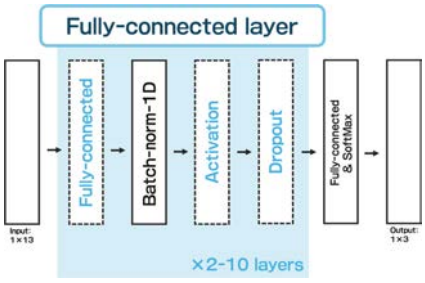

FIG. 2. Network structure of the MLP model.

the figure are targets to be optimized, according to the range of possible values for the design variables. The SoftMax function is used in the output layer to convert the output values into probabilities value, making it suitable for the classification problem. Table 2 shows the hyperparameters of the MLP model and their respective possible value ranges.

Table 2. List of hyperparameters.

Name	Type	Range
Learning rate lr	Real	$1e^{-5} \leq lr \leq 1e^{-1}$
Dropout rate dr_x	Real	$0.0 \leq dr_x \leq 1.0$ $(x = 1, 2, ..., 9)$
Weight decay lr	Real	$1e^{-10} \leq lr \leq 1e^{-3}$
Number of fully connected layers ln	Integer	$2 \leq ln \leq 10$
Number of nodes at layer x nn_x	Integer	$60 \leq nn_x \leq 125$ $(x = 1, 2, ..., 9)$
Number of epochs $epoch$	Integer	$2 \leq epoch \leq 10$
Activation function fn_x	Category	fn_x = {ReLU, LeakyReLU, ELU, PReLU, RReLU}
Batch normalization layer bn_x	Category	bn_x = {True, False} $(x = 1, 2, ..., 9)$
Batch size $batch$	Category	$batch$ = {8, 16, 32, 64, 128}
Optimization methods opt	Category	opt = {SGD, Momentum, AdaGrad, RMSProp, Adam}

4.3. Optimization problem

4.3.1. Objective function. Since the wine classification problem is a three-class classification problem, the cross-entropy error function, which is an error function that can be used to evaluate the three-class classification of the MLP model, is adopted as the objective function:

$$F(x) = -\sum_k t_k \log y_k, \tag{24}$$

where x is a combination of design variables and y is the MLP output value. t is a vector in which correct labels are converted to 1 and the incorrect labels are converted to 0.

4.3.2. Design variables. The hyperparameters shown in Table 2 are taken as the design variables.

In the PSO algorithm, the design variables should be defined as continuous ones. When optimizing discrete and categorical hyperparameters, the continuous variables optimized by PSO are converted into discrete and categoral ones.

When optimizing discrete hyperparameters such as the fully connected layers ln, fully connected layer x and number of epochs *epoch*, the continuous values are optimized by PSO and then converted to discrete values by performing a discretization process that rounds the decimal parts of the continuous design variables to integer values.

When optimizing categorical hyperparameters such as activation function fn_x, batch normalization layer bn_x, batch size *batch*, and optimization methods *opt*, continuous design variables are discretized and the discrete values are treated as indices for the elements of the categorical hyperparameters.

5. Numerical results

5.1. Effect of weight coefficients for search performance

The search performance of PSO algorithms strongly depends on the weigh coefficients c_1, c_2, c_3, and c_4. The effect of these weigh coefficients on the search performance is discussed here.

Some parameters are fixed as shown in Table 3. The maximum number of generations is specified as $t_{\max} = 100$ and the number of particles is set to $N = 30$ and 100. Simulations are performed 10 times with different parameters and the average values of the obtained results are compared.

Table 3. Fixed parameters for PSO algorithms.

Name	Value	Description
$t_{\max}$	100	Maximum number of generations
N	30, 100	Number of particles
$w_{\max}$	0.9	Maximum value of the inertia term for PSO-w
$w_{\min}$	0.4	Minimum value of the inertia term for PSO-w
N_P	5	Number of neghboring particles for local PSO
u	0.5	Parameter for UPSO

The weight parameters with the best search performance at $N = 30$ and 100 are determined by random search, which is summarized in Tables 4 and 5, respectively. The best parameters for PSO-cf, local PSO-cf and UPSO are similar in both cases. But, the others are different. It is concluded that the adequate parameters depend on the PSO algorithms.

TABLE 4. Best weight parameters for PSO algorithms at $N = 30$.

Algorithm	C1	C2	C3	C4
Basic PSO	2.0	2.0	–	–
PSO-w	0.04	1.96	–	–
PSO-cf	2.02	2.02	–	–
Local PSO-w	0.0	2.0	–	–
Local PSO-cf	2.01	2.01	–	–
UPSO	2.02	2.02	–	–
SG-PSO	0.02	1.98	0.5	–
SP-PSO	0.5	1.5	–	1.0

TABLE 5. Best weight parameters for PSO algorithms at $N = 100$.

Algorithm	C1	C2	C3	C4
Basic PSO	0.5	1.5	–	–
PSO-w	0.51	1.5	–	–
PSO-cf	2.02	2.02	–	–
Local PSO-w	0.49	1.5	–	–
Local PSO-cf	2.02	2.02	–	–
UPSO	2.05	2.04	–	–
SG-PSO	0.5	0.5	1.0	–
SP-PSO	1.0	1.0	–	1.0

5.2. Comparison of fitness convergence

The convergence histories of the fitness values at $N = 30$ are shown in Figs. 3 and 4, and for $N = 100$ in Figs. 5 and 6. The figures are plotted with the generation as the horizontal axis and the fitness values of the best particles

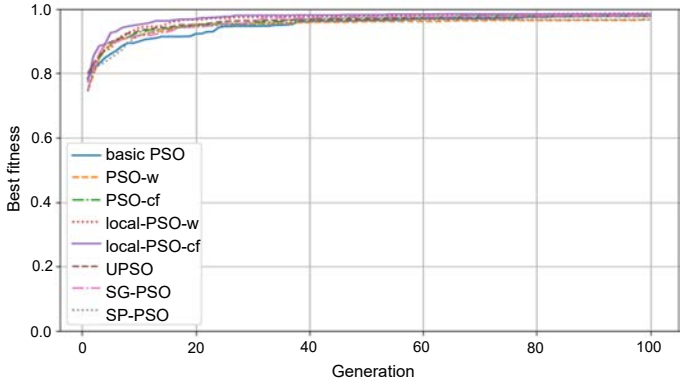

FIG. 3. Convergence histories of best fitness values at $N = 30$.

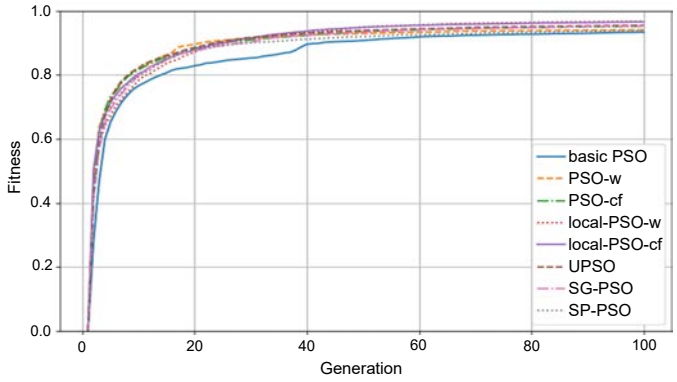

FIG. 4. Convergence histories of average fitness values at $N = 30$.

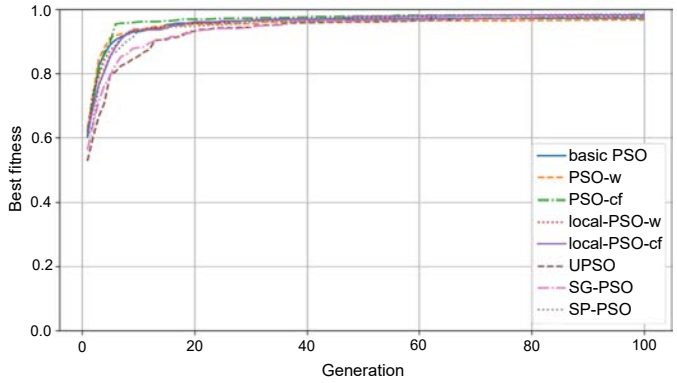

FIG. 5. Convergence histories of best fitness values at $N = 100$.

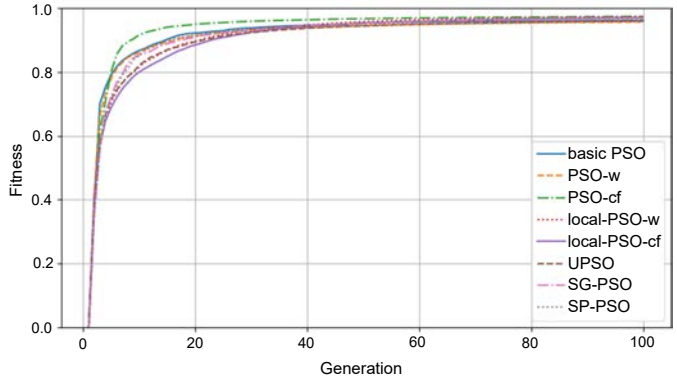

FIG. 6. Convergence histories of average fitness values at $N = 100$.

as the vertical axis. When comparing figures at $N = 30$ and 100, it is noticed that the figures for $N = 100$ converge faster than those for $N = 30$. Figure 3 indicates that, at $N = 30$, local PSO-cf and local PSO-w show the faster convergence than the others. Besides, Fig. 5 indicates that at $N = 100$, PSO-cf shows

the fastest convergence than the others. Therefore, it is concluded that PSO-cf and local PSO-cf are attractive from the perspective of convergence.

5.3. Comparison of classification accuracy trends

The number of particles is set to $N = 100$. Figure 7 shows the average and standard deviation of the classification accuracy calculated by each particle for each generation in 10 simulations. The figure is plotted with the classification accuracy on the vertical axis and the number of generations on the horizontal axis. The error bars in the graph represent the standard deviation. For all

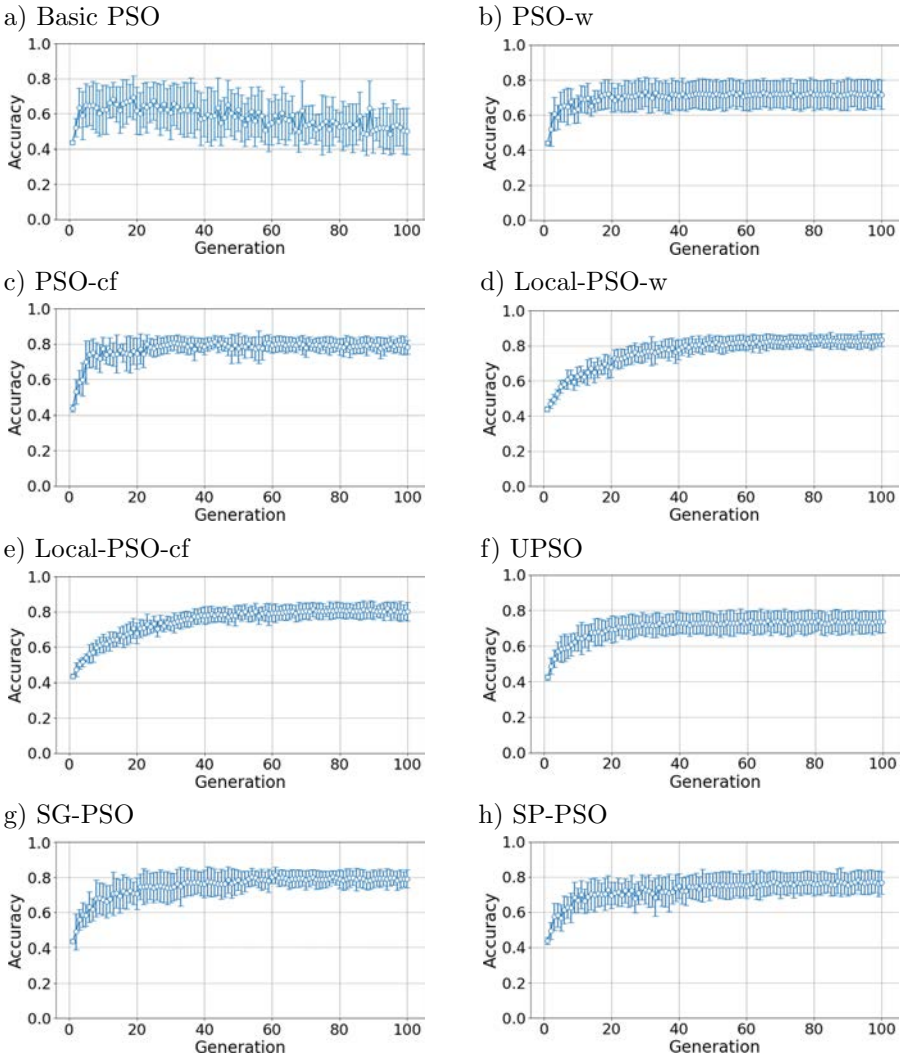

FIG. 7. Comparison of convergence histories of fitness.

PSO methods, the improvement in classification accuracy slows down after 10 to 20 generations, and the change in classification accuracy levels off after about 40 generations, indicating that particles converge to a specific solution at an early stage of the process.

5.4. Comparison of optimized hyperparameters

The optimized hyperparameters are compared. The number of particles is $N = 100$. PSO algorithms, random search, TPE and CMA-ES are used for searching the hyperparameters.

The hyperparameters are determined from the best particles that are finally obtained in 10 simulations. They are summarized in Tables 6, 7, 8, and 9, respectively. Table 6 also shows the fitness which is estimated as the average value of the fitness of the best particles from 10 simulations with the determined parameters.

Table 6 shows that the fitness of PSO algorithms is better than random search, TPE and CMA-ES. The best fitness value in PSO algorithms is evaluated as 0.974, achived by PSO-cf. The best fitness value among random search, TPE and CMA-ES is 0.878, obtained by random search. The best fitness value of PSO-cf is about 10% higher than that of random search.

Table 6. Optimized hyperparameters for the MLP (1).

Algorithm	Fitness	Learning rate	Dropout rate	Weight decay
Basic PSO	0.961	0.0121	0.0	1.00×10^{-05}
PSO-w	0.957	0.0148	0.0	1.00×10^{-05}
PSO-cf	0.974	0.1	0.0	1.00×10^{-05}
Local-PSO-w	0.973	0.0885	0.0	1.00×10^{-05}
Local-PSO-cf	0.969	0.0536	0.0	1.00×10^{-05}
UPSO	0.959	0.0017	0.0	1.00×10^{-05}
SG-PSO	0.968	0.0270	0.0	1.00×10^{-05}
SP-PSO	0.964	0.0728	0.0	1.00×10^{-05}
Random search	0.878	0.0255	0.321	0
TPE	0.849	0.0074	0.120	0
CMA-ES	0.793	0.0203	0.044	0

Table 6 also shows the learning rate, dropout rate and weight decay. As for the 'learning rate', all algorithms selected different values. However, the algorithms with high fitness values, such as PSO-cf, local-PSO-w, chose a larger learning rate than the others. For the 'dropout rate', all PSO algorithms selected the same value. For the 'weight decay', all algorithms selected the same value.

Table 7 shows the number of epochs, batch size, and optimization method. Except for UPSO, the other algorithms take similar values of 9 or 10. For the 'batch size', PSO algorithms selected either 8 or 16, which is similar to random search, TPE and CMA-ES. Similarly, for the 'optimization method', AdaGrad, RMSProp and Adam were selected.

Table 7. Optimized hyperparameters for the MLP (2).

Algorithm	Number of epochs	Batch size	Optimization method
Basic PSO	10	8	AdaGrad
PSO-w	10	8	RMSProp
PSO-cf	10	8	AdaGrad
Local-PSO-w	10	8	AdaGrad
Local-PSO-cf	10	8	RMSProp
UPSO	7	8	Adam
SG-PSO	10	16	RMSProp
SP-PSO	10	8	AdaGrad
Random search	9	8	RMSProp
TPE	9	16	RMSProp
CMA-ES	9	16	AdaGrad

Table 8 shows the number of fully connected layers and the number of nodes at each layer. Except for basic PSO and UPSO, the other PSO algorithms selected simpler network as evidenced by the number of fully connected layers being 2 or 3.

Table 8. Optimized hyperparameters for the MLP (3).

Algorithm	Number of fully connected layers	Number of nodes at layer
Basic PSO	8	[85, 65, 60, 67, 81, 91, 67, 60]
PSO-w	2	[60, 103]
PSO-cf	2	[60, 60]
Local-PSO-w	2	[125, 125]
Local-PSO-cf	2	[60, 125]
UPSO	6	[60, 114, 125, 63, 110, 110]
SG-PSO	2	[125, 60]
SP-PSO	2	[125, 60]
Random search	2	[90, 99]
TPE	2	[89, 62]
CMA-ES	3	[102, 107, 95]

Table 9 shows the activation function and batch normalization layer. ELU, ReLU and LeakyReLU are selected as the activation function while the batch normalization layer was used in some algorithms.

TABLE 9. Optimized hyperparameters for the MLP (4).

Algorithm	Activation function	Batch normalization layer
Basic PSO	ELU	TRUE
PSO-w	ELU	TRUE
PSO-cf	LeakyReLU	TRUE
Local-PSO-w	ReLU	TRUE
Local-PSO-cf	ELU	TRUE
UPSO	ELU	TRUE
SG-PSO	ReLU	TRUE
SP-PSO	ELU	TRUE
Random search	LeakyReLU	TRUE
TPE	ReLU	TRUE
CMA-ES	LeakyReLU	TRUE

6. Conclusion

PSO is a type of swarm intelligence optimization method inspired by the group behavior of living organisms. In addition to the basic PSO, which is the most basic algorithm, PSO methods include PSO-w, PSO-cf, local-PSO-w, local-PSO-cf, UPSO, SG-PSO, SP-PSO and so on. In this study, a PSO method was applied to the hyperparameter optimization problem of an MLP model. While traditional PSO deals only with continuous variables, the hyperparameters of an MLP model may take discrete or categorical variables. Hyperparameter optimization experiments were performed by converting continuous values into discrete or categorical variables. Random search, TPE, and CMA-ES were employed as comparison methods. The wine dataset was adopted as the subject of the numerical experiments. The results showed that PSO-cf performed the best and local PSO-w performed the second best among the PSO algorithms. The set of hyperparameters detemined by the PSO algorithms was relatively similar. The velocity update rules of PSO-cf and PSO-w were very similar, although they were presented individually by different researchers. The effect of the inertia term in the velocity update rule is controlled by the weight w in PSO-w and the constriction factor K in PSO-cf. In the early stages of the search, both algorithms tend to search better solutions from a wide solution space. Since the fitness function has a complicated distribution, there exist many local optima in the solution space. It is considered that the control of the inertia term in the

velocity update rule is effective for escaping local solutions and finding the optimal solution. An important observation from the numerical results is that PSO algorithms could find better hyperparameters than random search, TPE, and CMA-ES. Thus, it can be concluded that PSO is suitable for the hyperparameter optimization problem of MLP model.

This study discussed the search performance of PSO algorithms for the hyperparameter optimization problem of the MLP model in the wine dataset alone. To assess the validity of PSO algorithms for this problem, we plan to apply them the other dataset problems. After that, we will extend the same approach to design neural network models with different network structures.

References

1. I. Goodfellow *et al.*, Generative adversarial networks, *Communications of the ACM*, **63**(11): 139–144, 2020, https://doi.org/10.1145/3422622.
2. V. Mnih *et al.*, Human-level control through deep reinforcement learning, *Nature*, **518**: 529–533, 2015, https://doi.org/10.1038/nature14236.
3. M. Ferrari Dacrema, P. Cremonesi, D. Jannach, Are we really making much progress? A worrying analysis of recent neural recommendation approaches, [in:] *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys '19)*, Association for Computing Machinery, New York, NY, USA, pp. 101–109, 2019, https://doi.org/10.1145/3298689.3347058.
4. J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, [in:] *Advances in Neural Information Processing Systems 24 (NIPS 2011)*, Vol. 24, pp. 2546–2554, 2011.
5. J. Snoek, H. Larochelle, R.P. Adams, Practical Bayesian optimization of machine learning algorithms, [in:] *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, Vol. 25, pp. 2951–2959, 2012.
6. M. Feurer, F. Hutter, *Hyperparameter Optimization*, [in:] F. Hutter, L. Kotthoff, J. Vanschoren [Eds.], Automated Machine Learning: Methods, Systems, Challenges, Chapter 1, pp. 3–33, Springer, Cham, 2019, https://doi.org/10.1007/978-3-030-05318-5_1.
7. R.Z. Cabada, H.R. Rangel, M.L.B. Estrada, H.M.C. Lopez, Hyperparameter optimization in CNN for learning-centered emotion recognition for intelligent tutoring systems, *Soft Computing*, **24**(10): 7593–7602, 2020, https://doi.org/10.1007/s00500-019-04387-4.
8. P. Singh, S. Chaudhury, B.K. Panigrahi, Hybrid MPSO-CNN: Multi-level particle swarm optimized hyperparameters of convolutional neural network, *Swarm and Evolutionary Computation*, **63**: 100863, 2021, https://doi.org/10.1016/j.swevo.2021.100863.
9. N.M. Aszemi, P.D.D. Dominic, Hyperparameter optimization in convolutional neural network using genetic algorithms, *International Journal of Advanced Computer Science and Applications*, **10**(6): 269–278, 2019, https://doi.org/10.14569/IJACSA.2019.0100638.
10. P. Ribalta Lorenzo, J. Nalepa, M. Kawulok, L. Sanchez Ramos, J. Ranilla, Particle swarm optimization for hyper-parameter selection in deep neural networks, [in:] *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17)*, Association for Computing Machinery, pp. 481–488, 2017, https://doi.org/10.1145/3071178.3071208.

11. P. Ribalta Lorenzo, J. Nalepa, L. Sanchez Ramos, J. Ranilla, Hyper-parameter selection in deep neural networks using parallel particle swarm optimization, [in:] *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '17)*, Association for Computing Machinery, pp. 1864–1871, 2017, https://doi.org/10.1145/3067695.3084211.

12. Y. Wang, H. Zhang, G. Zhang, cPSO-CNN: An efficient PSO-based algorithm for fine-tuning hyper-parameters of convolutional neural networks, *Swarm and Evolutionary Computation*, **49**: 114–123, 2019, https://doi.org/10.1016/j.swevo.2019.06.002.

13. D. Sarkar, T. Khan, F. Ahmed Talukdar, Hyperparameters optimization of neural network using improved particle swarm optimization for modeling of electromagnetic inverse problems, *International Journal of Microwave and Wireless Technologies*, **14**(10): 1326–1337, 2022, https://doi.org/10.1017/S1759078721001690.

14. J. Kennedy, R.C. Eberhart, Particle swarm optimization, [in:] *Proceedings of ICNN'95 – International Conference on Neural Networks*, Vol. 4, pp. 1942–1948, 1995, https://doi.org/10.1109/ICNN.1995.488968.

15. Y. Shi, R. Eberhart, A modified particle swarm optimizer, [in:] *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence*, Anchorage, AK, USA, pp. 69–73, 1998, https://doi.org/10.1109/ICEC.1998.699146.

16. A. Maleki, M. Ameri, F. Keynia, Scrutiny of multifarious particle swarm optimization for finding the optimal size of a PV/wind/battery hybrid system, *Renewable Energy*, **80**: 552–563, 2015, https://doi.org/10.1016/j.renene.2015.02.045.

17. Y. Sun, Z. Wang, B.J. van Wyk, Local and global search based PSO algorithm, [in:] Y. Tan, Y. Shi, H. Mo [Eds.], *Advances in Swarm Intelligence. ICSI 2013. Lecture Notes in Computer Science*, Vol. 7928, pp. 129–136, Springer, Berlin, Heidelberg, 2013, https://doi.org/10.1007/978-3-642-38703-6_15.

18. K.E. Parsopoulos, M.N. Vrahatis, UPSO: A unified particle swarm optimization scheme, [in:] *International Conference of Computational Methods in Sciences and Engineering (ICCMSE 2004)*, pp. 868–873, CRC Press, 2019.

19. Y.-B. Shin, E. Kita, Search performance improvement of particle swarm optimization by second best particle information, *Applied Mathematics and Computation*, **246**: 346–354, 2014, https://doi.org/10.1016/j.amc.2014.08.013.

20. S. Watanabe, F. Hutter, c-TPE: Tree-structured Parzen estimator with inequality constraints for expensive hyperparameter optimization, [in:] E. Elkind [Ed.], *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, *IJCAI-23*, pp. 4371–4379, International Joint Conferences on Artificial Intelligence Organization, 2023, https://doi.org/10.48550/arXiv.2211.14411.

21. Y. Mei, H. Wang, Covariance matrix adaptation evolution strategy assisted by principal component analysis, *arXiv*, 2021, https://doi.org/10.48550/arXiv.2105.03687.

22. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, The MIT Press, 2016.

Received September 1, 2024; revised version November 26, 2024; accepted December 10, 2024; published online February 6, 2025.