Journal of Industrial and Management Optimization
Vol. 21, No. 4, April 2025, pp. 2472-2489
doi:10.3934/jimo.2024179

NONLINEAR OPTIMIZATION VIA NOVEL NEURAL NETWORK METHODS

Jiao Teng and Ka Fai Cedric Yiu*
Department of Applied Mathematics, The Hong Kong Polytechnic University, Hong Kong

(Communicated by Bin Li)

Abstract. This paper introduces a novel approach for solving high-dimensional nonlinear optimization problems by integrating neural networks into the optimization process. The method leverages the capabilities of neural networks to efficiently handle complex, high-dimensional data and to approximate discrete numerical solutions of nonlinear optimization problems using continuous functions. By combining the nonlinear mapping ability of neural networks with iterative optimization algorithms, the proposed approach provides a superior method to solve nonlinear optimization problems. The paper demonstrates the adaptability of neural network-based solution methods in solving various nonlinear optimization problems and illustrates that the method can be applied to many different problem scenarios. The effectiveness and correctness of the proposed method are demonstrated through examples.

1. Introduction.

Nonlinear Optimization has a wide range of applications in engineering, economics, finance, physics, biology, and many other disciplines [12]. The problem consists of maximizing or minimizing a function of one or more variables such that these variables satisfy certain constraints in the form of equations or inequalities. In modern science and engineering areas, high-dimensional nonlinear optimization plays an important role [2, 9] in engineering optimization [10], signal processing [11], and computer vision [5]. Many real-world problems can be modeled as nonlinear optimization problems, and the complexity of the problems increases along with the amount of data. These problems may involve a large number of variables and constraints, and optimal solutions are required to achieve certain objectives and satisfy specific requirements.

Solving high-dimensional nonlinear programming problems is challenging due to several factors: (1) The search space grows exponentially with the dimension of the problem, making it difficult to find the optimal solution. (2) There are numerous local optimal solutions that require significant computational resources and time to find the global optimal solution. (3) Complex nonlinear properties require a large number of iterations and calculations, increasing the complexity of the solution. (4) The presence of various nonlinear constraints further complicates the solution

2020 *Mathematics Subject Classification.* 90C26, 68T10.
Key words and phrases. Nonlinear optimization problems; Neural networks.
This paper is supported by RGC Grant PolyU 15203923, the Hong Kong Polytechnic University (4-ZZPT, 1-WZ0E), and Research Centre for Quantitative Finance (1-CE03), the first author is also supported by the National Natural Science Foundation of China under Grant No.12161076.
*Corresponding author: Ka Fai Cedric Yiu.

process. (5) Accuracy and stability requirements for the underlying numerical algorithms in high-dimensional spaces are demanding, necessitating careful handling of numerical errors and instability.

There are many methods for solving high-dimensional optimization problems, including traditional gradient descent techniques and trust region methods. There are searching techniques such as Quasi-Newton acceleration method [18], Particle Swarm Optimization Algorithm (PSO) [8], Genetic Algorithm (GA) [14], Interior methods [4] and so on[1, 16]. Some methods are prone to falling into local optima, while some are sensitive to learning rates and high computational complexity. For some standard optimization problems, such as quadratic constrained quadratic programming (QCQP) [7], they are still lacking good solution techniques. The structure of the problem is often explored to develop special solution techniques. In doing so, existing algorithms need to be adapted to new problem structures with significant expertise involved, which could take a long development cycle in the process. This increases the cost of developing solution to the problem. In view of this, it is advantageous to find a more universal solution method for all types of problems.

To overcome the challenges encountered, neural networks provide an attractive option for solving nonlinear optimization problems due to their ability to handle complex and high-dimensional nonlinear relationships, and are well suited for real-world problems [13, 6]. Their strong fitting and generalization capabilities enable them to learn and identify patterns and regularities within data effectively, enhancing the optimization process. Furthermore, due to the sheer number of variables and constraints, searching for an optimal solution in a high-dimensional space becomes very difficult as the solution space grows exponentially with the dimension of the problem. Neural networks can efficiently cope with large-scale data and high-dimensional spaces, leveraging their parallel computing capabilities to accelerate optimization. Consequently, integrating neural networks into nonlinear optimization problems improves efficiency, accuracy, and introduces innovative approaches to solving complex problems.

In this paper, we propose a novel neural network structure to represent the solution space and take advantage of the training process to minimize the objective function by continuously adjusting parameters and ensuring that the final solution satisfies constraints. This approach combines the nonlinear mapping ability of neural networks with the iterative optimization process of algorithms, effectively addressing complex optimization problems. The main contribution of this article is summarized as follows:

1. Approximate solution space of nonlinear optimization problems via continuous functional representations, which enhances accuracy and applicability and also improves computational efficiency. This approach provides a superior method for solving nonlinear optimization problems.
2. Take advantage of the ability of neural networks to efficiently process large-scale, high-dimensional data, thus providing a powerful tool for solving the challenges posed by complex nonlinear optimization problems.
3. The solution based on neural networks has shown adaptability and robustness across diverse contexts of nonlinear optimization problems, illustrating the method's wide-ranging applicability.

4. A novel loss function has been proposed to optimize the balance between objectives and constraints, along with an efficient neural network training method that includes customized learning rates and regularization to enhance algorithm performance and model generalization.

The paper is structured as follows. In Section 2, we introduce a unified form of nonlinear optimization problems. In Section 3, we apply the penalty function approach to nonlinear optimization problems and convert them into finding continuous function approximations. We then utilize the proposed NN algorithm to solve the reformulated problems. In Section 4, examples are used to demonstrate the effectiveness and validity of the proposed NN method.

2. **Nonlinear optimization problems.**

Nonlinear optimization problems take the following general form:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x} \in \mathcal{F}, \\ & \mathbf{x} \in \mathbb{R}^n. \end{aligned}$$

The variables $\mathbf{x} = (x_1, \ldots, x_n)^T$ are referred as the decision variables, the function $f(\cdot) : \mathbb{R}^n \to \mathbb{R}$ is known as the objective function, and the goal is to minimize this function. The expression "$\mathbf{x} \in \mathcal{F}$" indicates that the decision variables $\mathbf{x}$ must belong to a specified set $\mathcal{F}$.

Many problems of maximizing efficiency based on existing resources or minimizing costs to achieve a certain goal take different formats. A canonical form of constrained optimization can be represented by

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0, i = 1, \ldots, \mathcal{I}, \\ & h_j(\mathbf{x}) = 0, j = 1, \ldots, \mathcal{E}. \end{aligned} \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n, f(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}, g_i(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}, h_j(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}$.

These functions are continuous and differentiable usually. The feasible region of this problem is the set

$$\mathcal{F} = \{\mathbf{x} : g_1(\mathbf{x}) \leq 0, \ldots, g_{\mathcal{I}}(\mathbf{x}) \leq 0, h_1(\mathbf{x}) = 0, \ldots, h_{\mathcal{E}}(\mathbf{x}) = 0\}.$$

Let $\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_{\mathcal{I}}(\mathbf{x})) : \mathbb{R}^n \to \mathbb{R}^{\mathcal{I}}, \mathbf{h}(\mathbf{x}) = (h_1(\mathbf{x}), \ldots, h_{\mathcal{E}}(\mathbf{x})) : \mathbb{R}^n \to \mathbb{R}^{\mathcal{E}}$. Then (1) can be rewritten as

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{x}) \leq 0 \\ & \mathbf{h}(\mathbf{x}) = 0 \\ & \mathbf{x} \in \mathbb{R}^n. \end{aligned} \tag{2}$$

If the objective function is quadratic, then (1) becomes a quadratic programming problem (QP):

$$\begin{aligned} \min_{\mathbf{x}} \quad & Q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top G\mathbf{x} + g^\top \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b}, \\ & C\mathbf{x} \leq \mathbf{d}, \end{aligned} \tag{3}$$

where G is a symmetric matrix, $G \in \mathbb{R}^{n\times n}, g \in \mathbb{R}^n$.

When it comes to real-world problems, the constraints may take various forms, including linear and non-linear constraints, as well as the distinction between equalities and inequalities. For the convenience of subsequent study, we will list the types of constraints and provide the specific form of the non-linear optimization problem being studied as

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & A\mathbf{x} \le \mathbf{b} \\ & \bar{A}\mathbf{x} = \bar{\mathbf{b}} \\ & C(\mathbf{x}) \le 0 \\ & \bar{C}(\mathbf{x}) = 0 \end{aligned} \tag{4}$$

where A, $\bar{A}$, $\mathbf{b}$, $\bar{\mathbf{b}}$ are matrices and vectors respectively, and $C(\mathbf{x})$, $\bar{C}(\mathbf{x})$ are nonlinear vector functions.

3. **The neural network-based optimization method.**

In this section, we introduce the neural network structure to represent solution spaces of optimization problems, and to formulate the optimization problem as minimization of the objective function. Then we can apply neural network training to update the weights and biases of a neural network model via backpropagation algorithms to minimize the objective function and to ensure constraint satisfaction.

3.1. **Approximate by piecewise continuous functions.**

We define a set of intervals $S = (S_1, S_2, ..., S_N)$, where each variable x_i is assigned to one of these intervals. The interval information of each interval S_i is represented by $\mathbf{x}_{S_i}$, which corresponds to the value x_i. Furthermore, we define the collective interval information for all intervals as $\mathbf{x}_S = (x_{S_1}, x_{S_2}, ..., x_{S_N})$, and utilize it as the network input. The L-layer neural network is defined as follows:

Input layer: $\mathbf{F}^0(\mathbf{x}_S) = \mathbf{x}_S$,

Hidden layers: $\mathbf{F}^\ell(\mathbf{x}_S) = \sigma^\ell\left(\mathbf{w}^\ell \mathbf{F}^{\ell-1}(\mathbf{x}_S) + \mathbf{b}^\ell\right), \quad \text{for} \quad 1 \le \ell \le L-1,$

Output layer: $\mathbf{F}^L(\mathbf{x}_S) = \mathbf{w}^L \mathbf{F}^{L-1}(\mathbf{x}_S) + \mathbf{b}^L$.

Here, the parameters need to be determined include $V = (W, \mathbf{b})$, $W = (\mathbf{w}^1, \mathbf{w}^2, ..., \mathbf{w}^L)$, $\mathbf{b} = (b^1, b^2, ..., b^L)$, $\mathbf{w}^\ell$ and $\mathbf{b}^\ell$, $\ell = 1, 2, ..., L$ represent the weight matrices and bias vectors, respectively. Nonlinear activation functions $\sigma^\ell, \ell = 1, 2, ..., L$ are used to model nonlinear components. The lth component of the neural network's output layercan be briefly described as

$$F_l(\mathbf{x}_S, V) := \sum_{k=1}^{N_L} w_{lk}^{(L)} \sigma^L \left(\sum_{j=1}^{N_{L-1}} w_{kj}^{(L-1)} \sigma^{L-1} \left(\cdots \sigma^1 \left(\sum_{i=1}^{N_1} w_{ji}^{(1)} \mathbf{x}_{S_i} + w_{j0}^{(1)} \right) \cdots \right) + w_{k0}^{(L-1)} \right) + w_{l0}^{(L)}. \tag{5}$$

The piecewise continuous function $\tilde{F}(\mathbf{x}_S, V, \mathbf{r})$ can then be used to approximate the optimal solution $\mathbf{x}$ of nonlinear optimization problems. The continuous function $\tilde{F}(\mathbf{x}_S, V, \mathbf{r})$ is stated as

$$\tilde{F}(\mathbf{x}_S, V, \mathbf{r}) = \sum_{l=1}^{N} r_l F_l(\mathbf{x}_S, V) \tag{6}$$

with

$$r_l(x) = \begin{cases} 1 & \text{if } x \in S_l \\ 0 & \text{if } x \notin S_l \end{cases} \tag{7}$$

where $\mathbf{r}$ is the vector consist of the N scalar parameters $r_1, r_2, ..., r_N$, $\tilde{F}(\mathbf{x}_S, V, \mathbf{r})$ represents a piecewise constant function, with a value of r_l for all states within the set S_l.

To simplify the notation without losing the clarity of the relationship between the network output and the training parameters, we denote $\mathbf{F}^L(\mathbf{x}_S, V)$ by $\mathbf{F}(\mathbf{x}_S, V)$ where $\mathbf{F}(\mathbf{x}_S, V) = (F_1(\mathbf{x}_S, V), F_2(\mathbf{x}_S, V), ..., F_N(\mathbf{x}_S, V))$, and $F_l(\mathbf{x}_S, V)$ in equation (5) refers to lth component of $\mathbf{F}(\mathbf{x}_S, V)$. From the network architecture, it can be seen that the properties of the function $\mathbf{F}(\mathbf{x}_S, V)$ are embedded within the structure of the neural network. By training the parameters of the neural network to optimize the function $\tilde{F}(\mathbf{x}_S, V, \mathbf{r})$, we can obtain the value of $\mathbf{x}$ at each desired point, such as $x_1 = F_1(\mathbf{x}_S, V)$, $x_2 = F_2(\mathbf{x}_S, V), \ldots, x_N = F_N(\mathbf{x}_S, V)$, as illustrated in Fig 1.

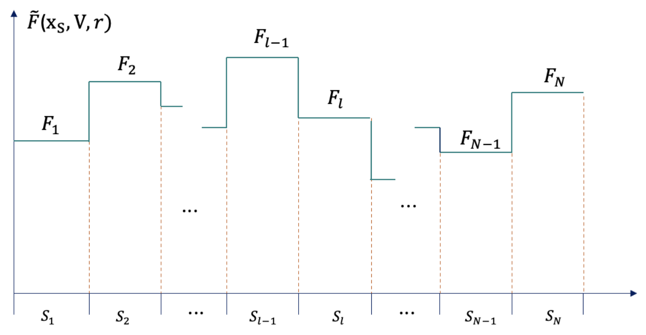


FIGURE 1. Approximate x by continuous functions $\tilde{F}(\mathbf{x}_S, V, \mathbf{r})$.

3.2. **Penalty approach.** In this section, we utilize the penalty function method to handle equality constraints and inequality constraints of the nonlinear programming problem. This approach involves transforming the constraints into the objective function. This simplifies the solution process by incorporating feasibility of the optimal solution as part of the objective function, and also ensures the convergence and stability of the algorithm.

For a general nonlinear optimization problem (2), the penalty function can be expressed as

$$\Phi(\mathbf{x}, \rho) = f(\mathbf{x}) + \rho_\mathbf{1} \sum_{i=1}^{\mathcal{I}} (\max(0, g_i(x)))^2 + \rho_\mathbf{2} \sum_{i=1}^{\mathcal{E}} h_i^2(x) \tag{8}$$

where $\rho_\mathbf{1}, \rho_\mathbf{2}$ are penalty factors for violating the constraints. For sufficiently large $\rho_\mathbf{1} > 0$ and $\rho_\mathbf{2} > 0$, the violation of the constraint function at the point of the minimum of the function $\Phi(\mathbf{x}, \rho)$ is small, and thus it can be regarded as an approximate optimal solution to problem (2). The terms $(\max(0, g(x)))^2$ and $h^2(x)$ correspond to the penalty term for the inequality constraint and the equality constraint, respectively.

For the unified form (4) studied in this paper, the penalty function $\Phi(\mathbf{x},\rho)$ can be expressed as

$$\Phi(\mathbf{x},\rho) = f(\mathbf{x}) + \rho_1\sum_{i=1}^{m}(\max(0,(\mathbf{a}_i\mathbf{x}-b_i)))^2 + \rho_2\sum_{j=1}^{n}(\bar{\mathbf{a}}_j\mathbf{x}-\bar{b}_j)^2 \tag{9}$$

$$+ \rho_3\sum_{k=1}^{p}|\bar{c}_k(\mathbf{x})|^2 + \rho_4\sum_{l=1}^{q}(\max(0,c_l(\mathbf{x})))^2 \tag{10}$$

where m,n,p,q denote the dimension of linear inequality constraints, linear equality constraints, nonlinear inequality constraints and nonlinear equality constraints, respectively, $\mathbf{a}_i$ is the ith row of A, $\bar{\mathbf{a}}_i$ is the ith row of $\bar{A}$, and ρ_1,ρ_2,ρ_3 and ρ_4 indicate penalty coefficients, b_i is the ith component of b, $\bar{b}_j$ is the jth component of $\bar{b}$.

After approximating $\mathbf{x}$ by $\mathbf{F}$, we map $\mathbf{x}$ into a cost approximation $\hat{\Phi}(\mathbf{F}(\mathbf{x}_S),\mathbf{r})$

$$\Phi(\mathbf{x},\rho) = \hat{\Phi}(\mathbf{F}(\mathbf{x}_S,V),\rho) \tag{11}$$

whose value depends on $\mathbf{F}$ and the parameter vector $\mathbf{r}$. Then, the nonlinear optimization problem (4) can be transformed into the following problem,

Problem. *Find function $\tilde{F}(\mathbf{x}_S,V,\mathbf{r})$, to minimize the following loss function*

$$\begin{aligned}\hat{\Phi}(\mathbf{x}_S,V,\mathbf{r}) = &f(\tilde{F}(\mathbf{x}_S,V,\mathbf{r})) + \rho_1\sum_{i=1}^{m}\max\left[0,\left(\mathbf{a}_i\cdot\tilde{F}(\mathbf{x}_S,V,\mathbf{r})-b_i\right)\right]^2\\ &+\rho_2\sum_{j=1}^{n}\left|\bar{\mathbf{a}}_j\cdot\tilde{F}(\mathbf{x}_S,V,\mathbf{r})-\bar{b}_j\right|^2\rho_3\sum_{k=1}^{p}\left|\bar{c}_k(\tilde{F}(\mathbf{x}_S,V,\mathbf{r})\right|^2\\ &+\rho_4\sum_{l=1}^{q}\max[0,c_l(\tilde{F}(\mathbf{x}_S,V,\mathbf{r})]^2\end{aligned} \tag{12}$$

Remark 3.1. In this context, it is important to distinguish between the roles of the two-parameter vectors. In particular, V parametrizes $\mathbf{F}(\mathbf{x}_S,V)$, which implies the features of the approximating function $\mathbf{F}(\mathbf{x}_S,V)$, while ρ is the penalty factor, representing the degree of penalty for different constraints.

3.3. Optimization by Neural Network Algorithms.

In this section, we choose the parameter vector V to determine the function $\tilde{F}(\mathbf{x}_S,V,\mathbf{r})$ by training $\hat{\Phi}(\mathbf{x}_S,V,\mathbf{r})$. A neural network architecture offers a set of parameterized functions $\tilde{F}(\mathbf{x}_S,V,\mathbf{r})$ in the form of (6) that can be utilized within the optimization framework described. As shown in Fig. 2, the decision variable x is approximated by a numerical vector of values $\tilde{F}(\mathbf{x}_S,V,\mathbf{r})$ with individual components $\mathbf{F}(\mathbf{x}_S,V)$. The linear output component of the hidden layer is used as the input to the nonlinear activation functions. After training with multiple hidden layers, $F_l(\mathbf{x},V)$ is generated at the output layer, and then the piecewise continuous function $\hat{\Phi}(\mathbf{x}_S,V,\mathbf{r})$ is generated with the parameter $\mathbf{r}$.

It is important to note that, unlike traditional supervised learning tasks using neural networks, in this case, we are using the neural network model as an optimizer. The output layer does not directly generate predictions or classification results, but instead continuously updates the variables to find the optimal solution through a back-propagation algorithm. Through neural network training, successive approximate solutions to nonlinear planning problems can be obtained.

Cybenko's approximation theorem [3] established that a feedforward neural network with at least one hidden layer of non-polynomial activation functions, such as sigmoid or ReLU, can approximate any continuous function to arbitrary accuracy. This theorem highlights the neural network's capacity for complex functional mapping. When we integrate a penalty term into the objective function to ensure that all constraints are considered during optimization, the neural network's weights are optimized using gradient descent, which iteratively refines the weights to minimize the objective function. In essence, the synergy of neural networks and gradient descent offers a versatile and potent framework for addressing optimization challenges. By leveraging neural networks to approximate the objective function and constraints, and employing gradient descent for weight optimization, this approach ensures convergence to an optimal solution that satisfies all constraints.

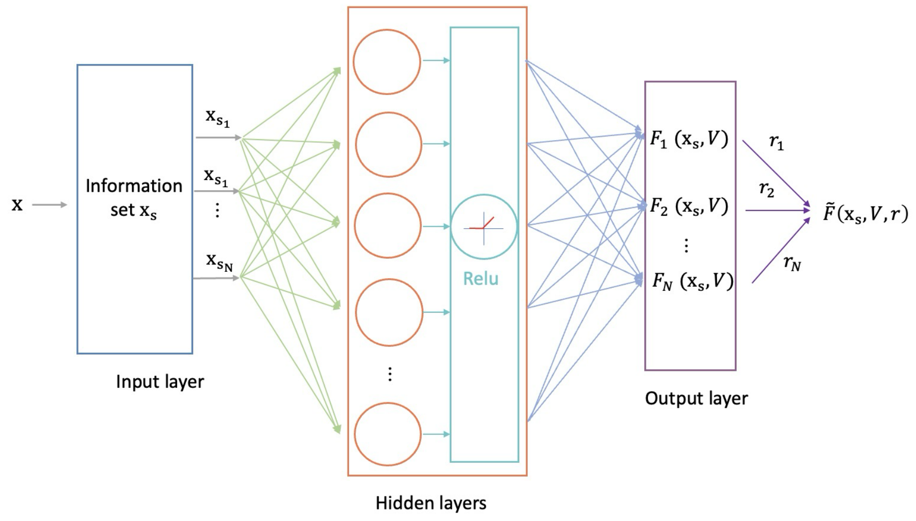


FIGURE 2. Approximate the decision variable $\mathbf{x}$ by a numerical vector of values $\tilde{F}(\mathbf{x}_S, V, \mathbf{r})$.

4. **Numerical results.**

In this section, we verify firstly the effectiveness of the proposed method to find the global optimal point of a nonlinear optimization problem using two test cases. Then, we tackle the QCQP problem and the nonlinear optimization problem with various constraints in both low and high-dimensional cases. The neural network-based optimization method example in this paper was implemented using Python, and the graphics card utilized was the GeForce RTX 4090, with the chip code AD102.

4.1. **Example 1 (2-dimensional Schubert function).**

The 2-dimensional Schubert function is a mathematical function defined on the 2-dimensional space. It is known for its complex and intricate pattern of local minima and maxima (as shown in Figure 3), making it a popular test function for optimization algorithms. The 2-dimensional Shubert function has 760 local minimas, 18 of which are global minimas with -186.73067 [17].

Algorithm 1

```
Require:
    $L_r$: Learning rate
    $N_T$: Number of training
    $N$: Dimension of the decision variable $\mathbf{x}$
    $m$: Dimension of linear equality constraints
    $n$: Dimension of linear inequality constraints
    $p$: Dimension of nonlinear equality constraints
    $q$: Dimension of nonlinear inequality constraints
    $layer, dense$: Number of layers of the neural network and neurons in the hidden layer
Ensure:
    Optimal solution
 1: Initialize the neural network model
 2: Initialize the optimizer with the given learning rate schedule
 3: Initialize the variable $\mathbf{x}$
 4: for $N_{epoh} = 1$ to $N_T$ do
 5:     Computing the cost function
 6:     Computing the loss function by adding the errors of each constraint function to the cost function
 7:     Calculating the gradient by backpropagation algorithm
 8:     Updating the network parameters using gradient descent algorithms to reduce the value of the loss function
 9:     Record the constraint values and objective value for analysis
10: end for
```

The 2-dimensional Schubert function is given by the following form, and the function's visual representations are depicted in Figure 3, which includes the front and top views.

$$\begin{gathered} f(x) = \left(\sum_{i=1}^{5} i \cos\left[(i+1)x_1 + i\right]\right)\left(\sum_{i=1}^{5} i \cos\left[(i+1)x_2 + i\right]\right), \\ -10 \leqslant x_i \leqslant 10, \quad i = 1, 2. \end{gathered} \tag{13}$$

As mentioned in the literature [15], the process of finding the global minimum using a simulated annealing algorithm typically necessitates approximately 100,000 function evaluations. In contrast, the neural networks- based optimization method only requires 100 training steps, taking a mere 4.4139 seconds, and yielding an optimal value of -186.73, with the optimal solution being -1.4249 and -0.8003. The variation of the objective function with the number of training steps is depicted in Figure 5. As can be seen from the figure, after repeated exploration, the value function eventually stabilized at -186.7305. This suggests that neural network based optimization methods can be effective in exploring globally optimal solutions.

4.2. **Example 2 (Levy function).** The Levy function's multi-peaked nature allows it to effectively test optimization algorithms in dealing with multi-peaked problems. Its numerous nonlinear properties also enable testing algorithm performance in nonlinear optimization problems. Additionally, Levy functions are suitable for high-dimensional spaces, making them valuable for evaluating algorithm performance in complex optimization situations.

$$f(\mathbf{x}) = \sin^2(\pi y_1) + \sum_{i=1}^{d-1} (y_i - 1)^2 \left[1 + 10\sin^2(\pi y_i + 1)\right] + (y_n - 1)^2 \left[1 + \sin^2(2\pi y_n)\right]$$
$$y_i = 1 + 0.25(x_i - 1), \quad -10 \leqslant x_i \leqslant 10, \quad i = 1, 2, \cdots, n. \tag{14}$$

We take the dimension of the decision variable to be 10000. The theoretical global optimum of the optimization problem corresponding to the levy function is 0 and the components of the optimal solution are 1. The optimal values and optimal solutions that we have calculated using the neural network approach are shown in Figure 6 and Figure 7. After 7.0015 seconds and 200 steps of neural network training, the objective function value was stabilized at 0 by continuously searching the solution space. This shows that neural network based optimization methods can be effective in exploring the global optimal solution in high-dimensional problems as well.

4.3. **Example 3 (Quadratically Constrained Quadratic Programming (QCQP)).** For this example, we consider the quadratically constrained quadratic programming problem (QCQP) with the following form:

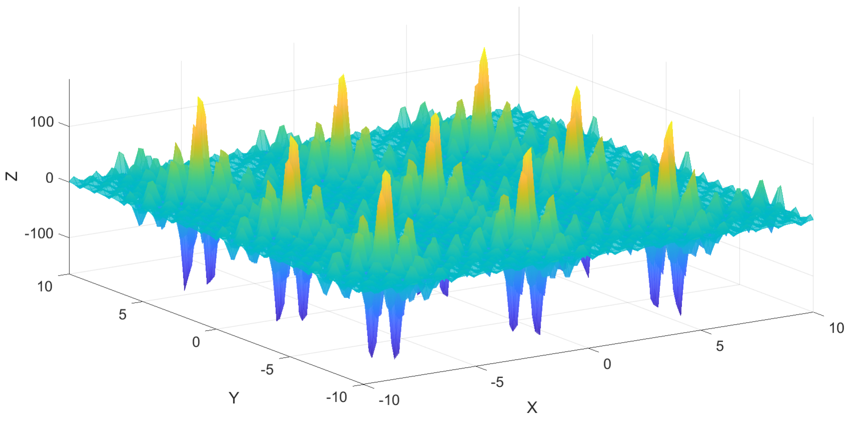
$$\begin{aligned} &\min \mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ &\text{s.t. } \mathbf{x}^T Q_c \mathbf{x} \leq \mathbf{b} \end{aligned} \tag{15}$$

where $\mathbf{x}$ is an n-dimensional vector, Q is an $n \times n$ symmetric matrix, $\mathbf{c}$ is an n-dimensional vector, Q_c is an $n \times n$ matrix, and $\mathbf{b}$ is an 1-dimensional vector.

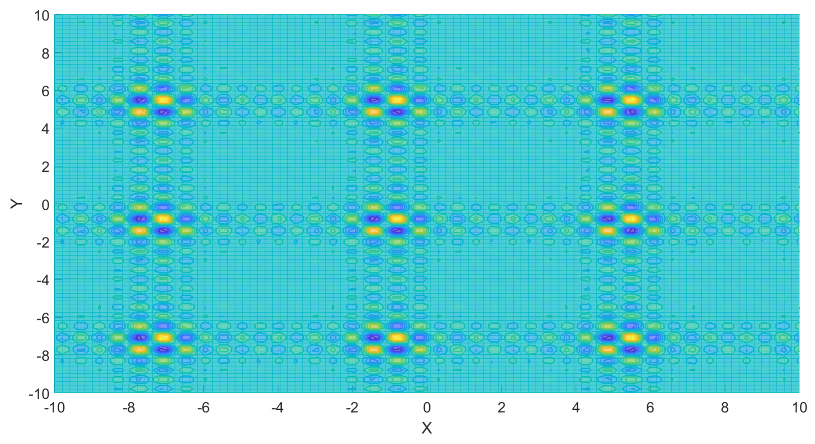
4.3.1. *Example 3.1 (QCQP - Low dimension).* There are relatively few algorithms available for solving Quadratically Constrained Quadratic Programming (QCQP) problems, and a generalized efficient algorithm is notably lacking. To illustrate the effectiveness of our proposed method in addressing QCQP challenges, this subsection begins by comparing two approaches: one utilizing neural networks and the other employing the matlab Fmincon toolbox.

In low-dimensional cases, we consider the dimension of the decision variable to be $N = 10$ and $N = 1000$. Matrices Q, Q_c and vectors $\mathbf{c}$ and $\mathbf{b}$ are randomly generated. We have repeated at least 3 times generating 3 different problems and compared well with the results from Matlab. One set of typical results for both examples are recorded in the table along with the number of network layer & dense. In Figure 8, to clearly show the decreasing trend of the training process, we created some subfigures on the larger plot. Each subgraph corresponds to a different stage of the optimization process, its color corresponds to the area indicated by the arrows on the axes. It can be seen from the subgraphs that, as the training goes deeper, the cost function value decreases and converges to the optimal value gradually. Figure 9 demonstrates the optimal solution for the variable x.

4.3.2. *Example 3.2 (QCQP - High dimension).* Compared to small-scale QCQP problems, large-scale problems face multiple challenges, including computational complexity, an increase in the number of local minima, and an increase in the difficulty of constraint satisfaction, leading to the fact that existing optimization



(a)



(b)

FIGURE 3. The images of 2-dimensional Schubert function.

Dimension	$N = 10$	$N = 1000$
Optimal objective function	NN: -0.3548 Fmincon: -0.35479	NN: -23.587 Fmincon: -23.587
Layer & Dense of NN	4, 8	4, 16
The number of training	15000	60000

TABLE 1. Comparative results of QCQP in low dimensions.

algorithms may not scale effectively. In this subsection, we demonstrate the effectiveness of our approach in scaling to large-scale problems.

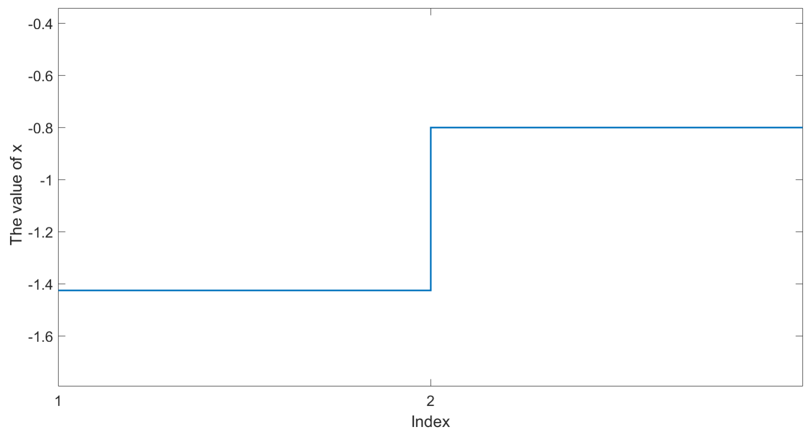


FIGURE 4. Optimal solution for Example 1.

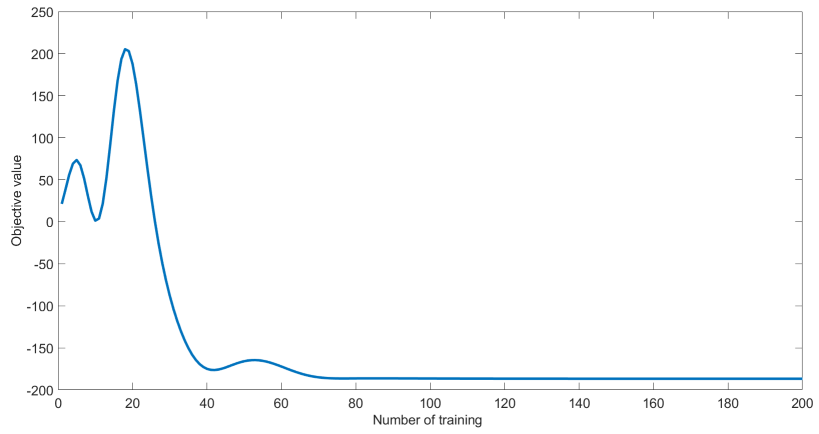


FIGURE 5. Objective value for Example 1.

In this example, we consider the dimension of the independent variable to be $N = 10000$ and $N = 30000$, the results for both examples are recorded in Table 2 along with the number of network layer & dense.

In Figure 10, we illustrate the variation of the objective function with $N = 10000$. To better observe its decreasing trend, we include several sub-figures within the main figure, with colors corresponding to the areas indicated by the arrows on the axes, from which we can see that the objective function gradually decreases and stabilizes at the optimal value of -226.539.

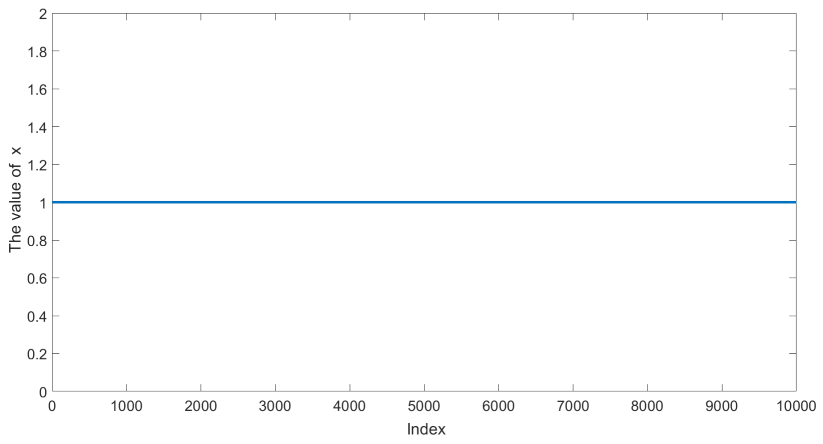


FIGURE 6. Optimal solution for Example 2.

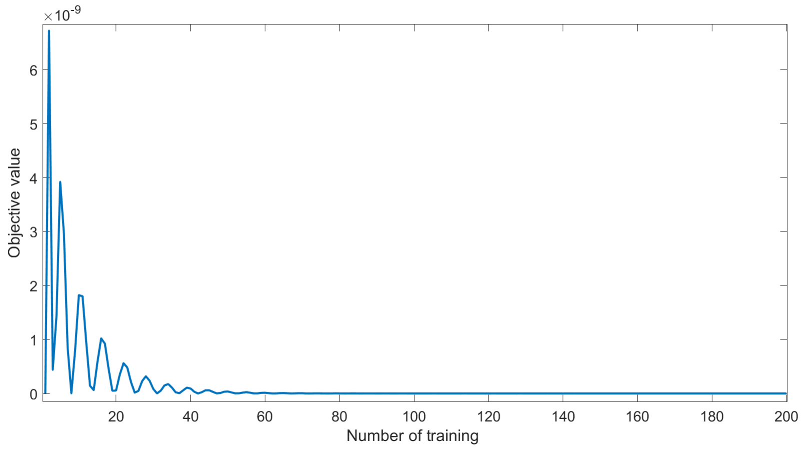


FIGURE 7. Objective value for Example 2.

Dimension	$N = 10000$	$N = 30000$
Optimal objective function	NN: -226.539 Fmincon: failed	NN: -655.842 Fmincon: failed
Layer & Dense of NN	16, 32	64, 128
The number of training	100000	300000

TABLE 2. Comparative results of QCQP in high dimensions.

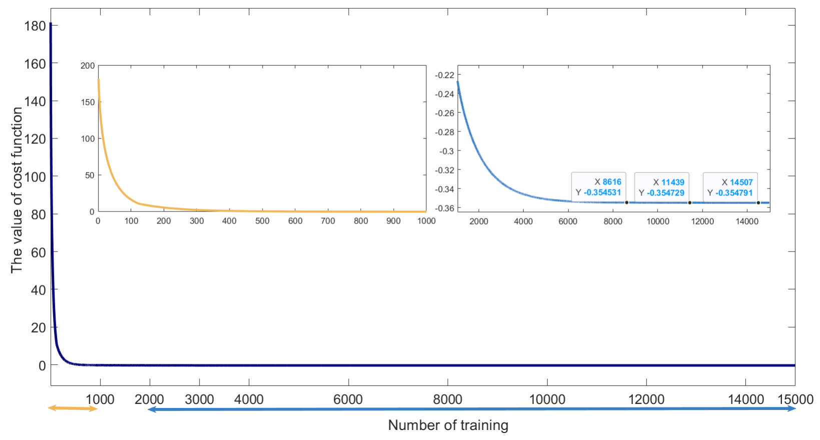


FIGURE 8. Cost function value of the NN for the 10-dimensional QCQP.

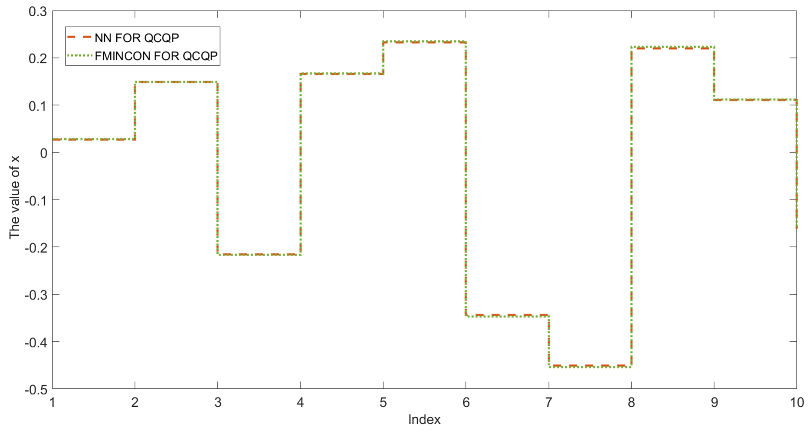


FIGURE 9. Optimal solutions of NN and Fmincon for the 10-dimensional QCQP.

4.4. **Example 4 (Nonlinear Optimization Problem (NLOP)).** In this section, we verify the ability of our algorithm for solving nonlinear optimization problems, and similar to the previous example, we compute the low and high dimensional problems separately.

4.4.1. *Example 4.1(Nonlinear Optimization Problem- Low dimension).* In this example, we focus on a nonlinear programming (NLOP) problem with a four-dimensional optimization variable x. The problem is subject to two linear equality and two linear

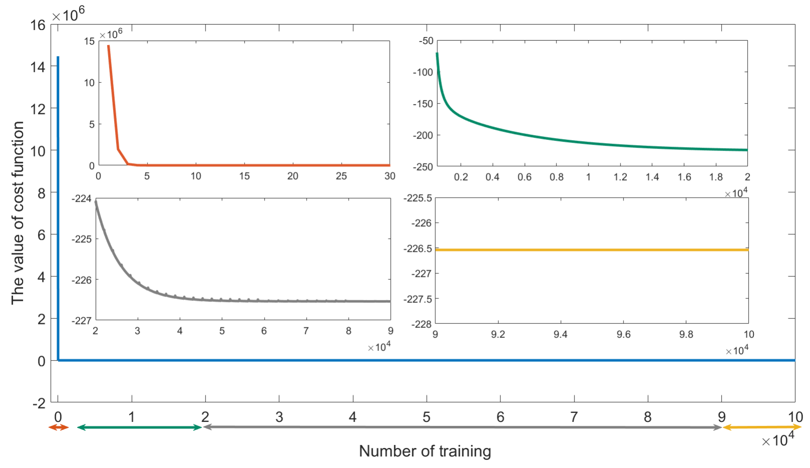


FIGURE 10. Cost function value of the NN for the 10000-dimensional QCQP.

inequality constraints. The problem is presented as follows:

$$\min f(x) = \sum_{i=1}^{4}(x_i^2 + 2sinx_i)$$
$$\text{s.t.} \quad \begin{array}{c} A\mathbf{x} \leq \mathbf{b} \\ B\mathbf{x} = \mathbf{d} \\ C(\mathbf{x}) = \sum_{i=1}^{4}(x_i^3 + |x_i|) \leq 20 \\ D(\mathbf{x}) = \sum_{i=1}^{4} x_i^2 = 12 \end{array} \tag{16}$$

where

$$A = \begin{bmatrix} 0.5085 & 4.8168 & 0.4205 & 1.5918 \\ 7.1162 & 0.6389 & 2.7053 & 0.6910 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 3.2040 \\ 3.1195 \end{bmatrix},$$
$$B = \begin{bmatrix} 0.5269 & 5.6120 & 0.8137 & 7.4510 \\ 1.8911 & 0.6409 & 12.0439 & 0.7832 \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} 2.1058 \\ 8.9273 \end{bmatrix}.$$

The network parameters employed are as follows, the network consists of 8 layers with 32 neurons in each hidden layer, and the optimization is performed using the Adagrad algorithm. The optimal solution values for x from both methods are presented in Table 3. The process of incorporating a penalty term for the constraints within the loss function ensures that the constraints are progressively met as the training advances. This gradual satisfaction of the constraints is clearly depicted in Figure 11, which illustrates the variation in the constraint function over the course of training.

Furthermore, Figure 12 presents a visual account of how the value function evolves throughout the training process. The figure is accompanied by two detailed plots that offer insights into specific aspects of the training: the first plot zooms in on the descent of the function during the training phase, while the second plot clearly demonstrates the stabilization of the value function, reaching a settled value of 14.361.

Index	NN	Fmincon
1	-3.0630	-3.0630
2	1.0151	1.0150
3	1.1940	1.1939
4	-0.3957	-0.3957

TABLE 3. Comparative results for low-dimensional NLOP.

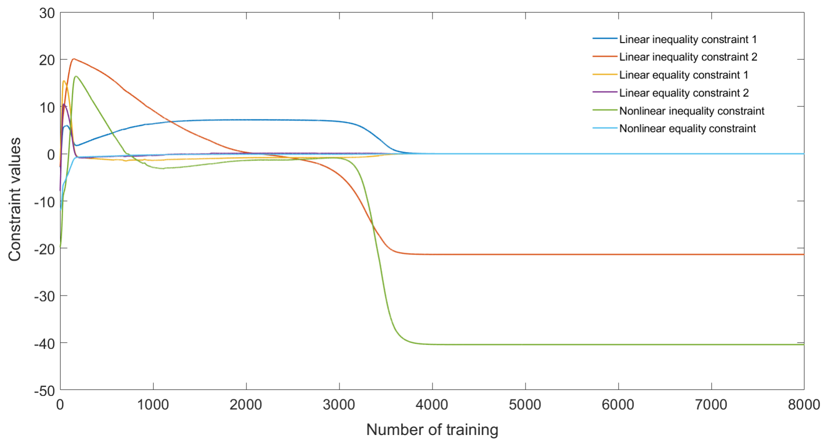


FIGURE 11. Constraint values for NLOP in low-dimensions.

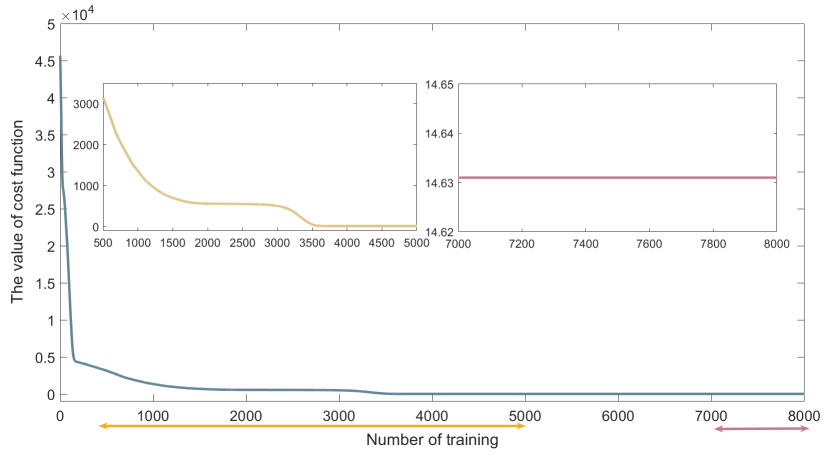


FIGURE 12. Optimal solutions of NN and Fmincon for NLOP in low dimensions.

4.4.2. *Example 4.2 (Nonlinear Optimization Problem - High dimension).*

$$\min f(\mathbf{x}) = \sum_{i=1}^{N}(x_i^2 + cosx_i)$$
$$\text{s.t.} \quad \begin{array}{c} A\mathbf{x} \leq \mathbf{b} \\ B\mathbf{x} = \mathbf{d} \\ C(\mathbf{x}) = \sum_{i=1}^{N}(x_i^3 + x_i sinx_i) \leq 600 \\ D(\mathbf{x}) = \sum_{i=1}^{N}|x_i| = 1000 \end{array} \tag{17}$$

In this example, we set $N = 10000$, and define 4 types of constraints, including 2 linear and 2 nonlinear constraints. The matrices A and B, which are both of dimensions $2 \times n$, as well as the vectors b and d, which are both 2×1, are generated randomly using Python. For the optimization process, the Adam optimizer is utilized during the initial 20% of the training steps, after which the Adadelta optimizer takes over for the remainder of the training. The neural network configuration consists of 32 layers with 128 neurons in each hidden layer. After 10000 pieces of training, the value function tends to stabilize to 10056.38.

Figure 13 presents a visual representation of how the constraints are satisfied throughout the training process. The accompanying smaller figure provides an enlarged view of certain details, with colors corresponding to the double-arrowed sections in the main figure. Figure 14 demonstrates the downward trend of the value function during the training process, showing how it eventually stabilizes. The color coding in this figure is consistent with that used in the previous figure for ease of comparison and understanding.

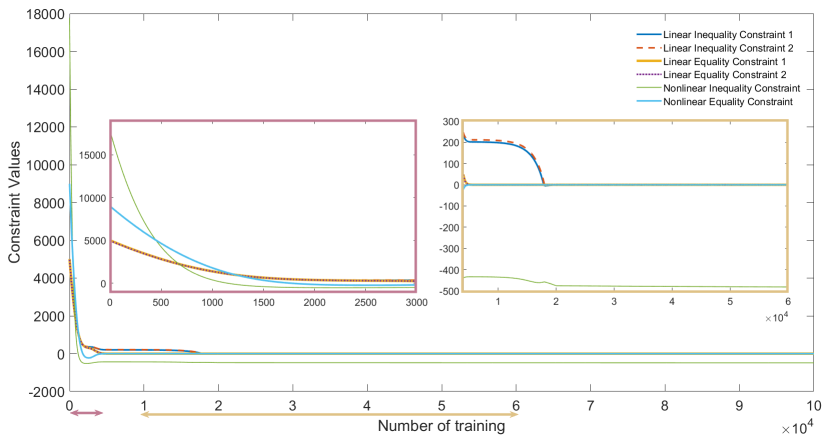


FIGURE 13. Constraint values for high-dimensional NLOP.

5. **Conclusion.** This paper presents a new approach for solving complex nonlinear optimization problems by incorporating neural networks into the optimization process. The proposed method exploits the ability of neural networks to handle high dimension and complex nonlinear relationships, and provides a promising solution

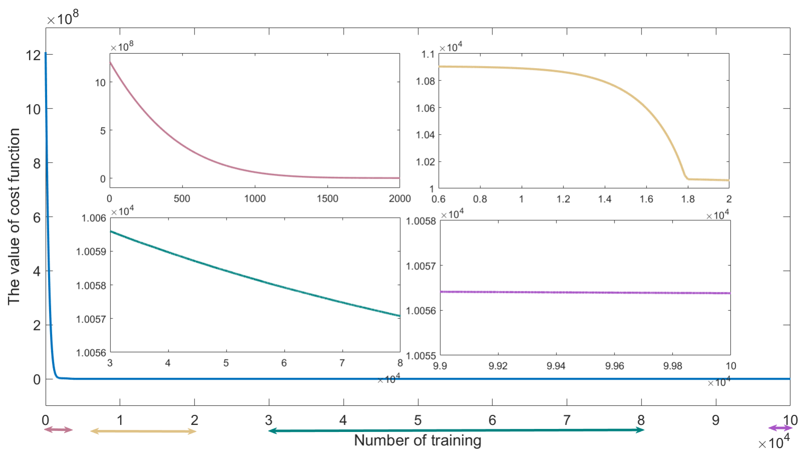


FIGURE 14. Cost function value for high-dimensional NLOP.

to the challenges posed by such problems by developing suitable neural network structures during the training process of optimization algorithms. This work explores new possibilities for solving challenging nonlinear optimization problems and demonstrates the potential for generalization to new and diverse problem solutions. Although our neural network-based optimization approach performs well in dealing with complex problems, it is possible that the training process can become expensive and time-consuming when faced with very large-scale problems with millions of variables and constraints. At the same time, the dynamic nature of real-time optimization problems and the frequent updating of constraints also place higher demands on the robustness and adaptivity of neural networks. These will be future research directions, including the development of more efficient training algorithms and neural network structures to reduce computational complexity, as well as the study of adaptive models to achieve fast responses to real-time changes, thus enhancing the applicability and robustness of our approach.

REFERENCES

[1] M. Z. Ali, N. H. Awad and P. N. Suganthan, Multi-population differential evolution with balanced ensemble of mutation strategies for large-scale global optimization, *Applied Soft Computing*, **33** (2015), 304-327.

[2] I. Aljarah, H. Faris and S. Mirjalili, Optimizing connection weights in neural networks using the whale optimization algorithm, *Soft Computing*, **22** (2018), 1-15.

[3] G. Cybenko, Approximation by superpositions of a sigmoidal function, *Mathematics of Control, Signals and Systems*, **2** (1989), 303-314.

[4] A. Forsgren, P. E. Gill and M. H. Wright, Interior methods for nonlinear optimization, *SIAM Review*, **44** (2002), 525-597.

[5] T. Georgiou, Y. Liu, W. Chen and M. Lew, A survey of traditional and deep learning-based feature descriptors for high dimensional data in computer vision, *International Journal of Multimedia Information Retrieval*, **9** (2020), 135-170.

[6] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, Adapt. Comput. Mach. Learn, MIT Press, Cambridge, MA, 2016.

[7] K. Huang and N. D. Sidiropoulos, Consensus-ADMM for general quadratically constrained quadratic programming, *IEEE Transactions on Signal Processing*, **64** (2016), 5297-5310.

[8] D. L. Jia, G. X. Zheng, B. Y. Qu and M. K. Khan, A hybrid particle swarm optimization algorithm for high-dimensional problems, *Computers & Industrial Engineering*, **61** (2011), 1117-1122.

[9] S. Mahdavi, M. E. Shiri and S. Rahnamayan, Metaheuristics in large-scale global continues optimization: A survey, *Information Sciences*, **295** (2015), 407-428.

[10] Y. S. Ong, P. B. Nair, A. J. Keane and K. W. Wong, Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems, *Knowledge Incorporation in Evolutionary Computation*, **167** (2005), 307-331.

[11] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura and P. Vandergheynst, Graph signal processing: Overview, challenges, and applications, *Proceedings of the IEEE*, **106** (2018), 808-828.

[12] P. M. Pardalos, H. E. Romeijn and H. Tuy, Recent developments and trends in global optimization, *Journal of computational and Applied Mathematics*, **124** (2000), 209-228.

[13] J. Schmidhuber, Deep learning in neural networks: An overview, *Neural networks*, **61** (2015), 85-117.

[14] J.-T. Tsai, T.-K. Liu and J.-H. Chou, Hybrid Taguchi-genetic algorithm for global numerical optimization, *IEEE Transactions on Evolutionary Computation*, **8** (2004), 365-377.

[15] K. F. C. Yiu, Y. Liu and K. L. Teo, A hybrid descent method for global optimization, *Journal of Global Optimization*, **28** (2004), 229-238.

[16] J. Yuan, D. Yang, D. Xun, K. Teo, C. Wu, A. Li, Z. Gong, K. Qu and K. Gao, Sparse optimal control of cyber-physical systems via PQA approach, to appear, *Pacific Journal of Optimization*.

[17] M. Zbigniew, Genetic algorithms+ data structures= evolution programs, Second edition, Springer-Verlag, Berlin, 1994.

[18] H. Zhou, D. Alexander and K. Lange, A quasi-Newton acceleration for high-dimensional optimization algorithms, *Statistics and Computing*, **21** (2011), 261-273.

Received April 2024; revised October 2024; early access November 2024.