Using Neural Networks as Surrogate Models in Differential Evolution Optimization of Truss Structures

Tran-Hieu Nguyen(✉) and Anh-Tuan Vu

National University of Civil Engineering, Hanoi, Vietnam
hieunt2@nuce.edu.vn

Abstract. In this study, Differential Evolution, a powerful metaheuristic algorithm, is employed to optimize the weight of truss structures. One of the major challenges of all metaheuristic algorithms is time-consuming where a large number of structural analyses are required. To deal with this problem, neural networks are used to quickly evaluate the response of the structures. Firstly, a number of data points are collected from a parametric finite element analysis, then the obtained datasets are used to train neural network models. Secondly, the trained models are utilized to predict the behavior of truss structures in the constraint handling step of the optimization procedure. Neural network models are developed using Python because this language supports many useful machine learning libraries such as scikit-learn, tensorflow, keras. Two well-known benchmark problems are optimized using the proposed approach to demonstrate its effectiveness. The results show that using neural networks helps to greatly reduce the computation time.

Keywords: Structural optimization · Truss structure · Differential evolution · Machine learning · Neural network · Surrogate model

1 Introduction

Truss structures have been widely used in large-span buildings and constructions due to their advantages as lightweight, robustness, durability. However, because truss structures are intricate and complex with many individual elements, a good design requires a lot of human resources. The conventional process to design truss structures is the "trial and error" method where the result strongly depends on the designer's experience. Moreover, for large-scale trusses with a wide list of available profiles, the number of candidates becomes too prohibitive. For example, a simple 10-bar truss in which the cross-section of each member must be selected from a set of 42 available profiles has totally 42^{10} possible options [1]. In such cases, the "trial and error" method is impossible. In order to help a designer find the best choice, another approach called optimization-based design method was invented and has been constantly developed.

In recent decades, many optimization algorithms have been proposed and successfully applied to solve structural problems. Several algorithms are designed based on natural evolution such as Genetic Algorithm (GA) [1], Evolution Strategy (ES) [2],

N. T. Nguyen et al. (Eds.): ICCCI 2020, LNAI 12496, pp. 152–163, 2020.
https://doi.org/10.1007/978-3-030-63007-2_12

Differential Evolution (DE) [3]. Some other algorithms are inspired by the swarm intelligence like Particle Swarm Optimization (PSO) [4], Ant Colony Optimization (ACO) [5], Artificial Bee Colony (ABC) [6], etc. These algorithms use an effective "trial and error" method to find the minimum. In this way, the global optimum is usually found, but it requires a huge amount of function evaluation. This is further complicated for structural optimization problems where the function evaluation process is usually performed by the finite element analysis (FEA). To demonstrate this, the duration of a single analysis for the tied-arch bridge which consists of 259 members described in [7] is approximately 60 s but the optimization time is nearly 133 h. In recent years, the development of machine learning (ML) has shown a promising solution to this challenge. By building ML-based surrogate models to approximately predict the behavior of structures, the number of FEAs is greatly reduced and the overall computation time is consequently decreased.

The idea of using ML as surrogate models in structural optimization is not new. Many related studies have been previously published [8–11]. However, due to the recent development in the field of ML, this topic needs to be carefully considered. Among these evolutionary algorithms, DE is preferred because of its advantages [12]. On the other hand, while many ML algorithms exist, the capacity of the NN has been proved [13]. Therefore, an optimization approach that combines NN and DE is proposed in this study. The paper consists of five main sections. The background theory is briefly presented in Sect. 2. Based on the integration between NN and DE, an optimization approach is proposed in Sect. 3. Two computational experiments are performed in Sect. 4. The obtained results are compared with those of other algorithms. Finally, some main conclusions are pointed out in Sect. 5.

2 Background Theory

2.1 Optimization Problem Statement

The weight optimization problem of a truss structure can be stated as follows.

Find a vector of $\mathbf{A}$ representing the cross-sectional areas of truss members:

$$\mathbf{A} = [A_1, A_2, \ldots, A_n] \text{ where } A_i^{min} \le A_i \le A_i^{max} \tag{1}$$

$$\text{to minimize :} \qquad W(\mathbf{A}) = \sum_{i=1}^{n} \rho_i A_i l_i \tag{2}$$

$$\text{subject to:} \qquad \begin{cases} g_{i,s} = \sigma_i \big/ \sigma_i^{allow} - 1 \le 0 \\ g_{j,u} = \delta_j \big/ \delta_j^{allow} - 1 \le 0 \end{cases} \tag{3}$$

where: ρ_i, A_i, l_i are the density, the cross-sectional area and the length of the i^{th} member, respectively; A_i^{min}, A_i^{max} are the lower and upper bounds for the cross-sectional area of the i^{th} member; $\sigma_i, \sigma_i^{allow}$ are the actual stress and the allowable stress of the i^{th} member; n is the number of members; δ_j, δ_j^{allow} are the actual and the allowable displacements of the j^{th} node; m is the number of nodes.

2.2 Differential Evolution Algorithm

DE was originally proposed by K. Price and R. Storn in 1997 [14]. Like many evolutionary algorithms, the DE comprises four basic operators namely initialization, mutation, crossover, and selection. They are briefly described as follows.

Initialization: a random population of *Np* individuals is generated. Each individual which is a *D*-dimensional vector represents a candidate of the optimization problem. *Np* is the number of candidates in the population and *D* is the number of design variables. The j^{th} component of the i^{th} vector can be generated using the following express:

$$x_{ij}^{(0)} = x_j^{\min} + rand[0,1]\left(x_j^{\max} - x_j^{\min}\right) \tag{4}$$

where: *rand*[0,1] is a uniformly distributed random number between 0 and 1; x_j^{min} and x_j^{max} are the lower and upper bound for the j^{th} component of the i^{th} vector.

Mutation: a mutant vector $\mathbf{v}_i^{(t)}$ is created by adding a scaled difference vector between two randomly chosen vectors to a third vector. Many different mutation strategies have been proposed like 'DE/rand/1/', 'DE/rand/2', 'DE/best/1', 'DE/best/2', etc. In this study, a powerful mutation strategy, called 'DE/target-to-best/1' is employed. This variant produces the mutant vector based on the best individual of the population $\mathbf{x}_{best}^{(t)}$ and the target individual $\mathbf{x}_i^{(t)}$ as follows:

$$\mathbf{v}_i^{(t)} = \mathbf{x}_i^{(t)} + F \times \left(\mathbf{x}_{best}^{(t)} - \mathbf{x}_i^{(t)}\right) + F \times \left(\mathbf{x}_{r1}^{(t)} - \mathbf{x}_{r2}^{(t)}\right) \tag{5}$$

where: $r_1 \neq r_2$ are randomly selected between 1 and *Np*; *F* is the scaling factor.

Crossover: a trial vector $\mathbf{u}_i$ is created by getting each variable value from either the mutant vector $\mathbf{v}_i$ or the target vector $\mathbf{x}_i$ according to the crossover probability.

$$u_{ij}^{(t)} = \begin{cases} v_{ij}^{(t)} \text{ if } j = K \text{ or } rand[0,1] \leq Cr \\ x_{ij}^{(t)} \text{ otherwise} \end{cases} \tag{6}$$

where: $u_{ij}^{(t)}$, $v_{ij}^{(t)}$ and $x_{ij}^{(t)}$ are the j^{th} component of the trial vector, the mutant vector, and the target vector, respectively; *K* is any random number in the range from 1 to *D*, this condition ensures that the trial vector differs the target vector by getting at least one mutant component; *Cr* is the crossover rate.

Selection: for each individual, the trial vector is chosen if it has a lower objective function value; otherwise, the target vector is retained. The selection operator is described as follows:

$$\mathbf{x}_i^{(t+1)} = \begin{cases} \mathbf{u}_i^{(t)} \text{ if } f\left(\mathbf{u}_i^{(t)}\right) \leq f\left(\mathbf{x}_i^{(t)}\right) \\ \mathbf{x}_i^{(t)} \text{ otherwise} \end{cases} \tag{7}$$

where: $f(\mathbf{u}_i^t)$ and $f(\mathbf{x}_i^t)$ are the objective function value of the trial vector and the target vector, respectively.

For constrained problems, the mutation and crossover operators could produce infeasible candidates. In such case, the selection is based on three criteria: (i) any feasible

candidate is preferred to any infeasible candidate; (ii) among two feasible candidates, the one having lower objective function value is chosen; (iii) among two infeasible candidates, the one having lower constraint violation is chosen [15]. For the implementation, the objective function is replaced by the penalty function:

$$f'(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{n} r_i \times \max(0, g_i(\mathbf{x})) \tag{8}$$

where: r_i is the penalty parameter; g_i is the constraint violation.

The initialization operator is performed only one time at the initial iteration while three last operators are repeated until the termination criterion is satisfied.

2.3 Neural Networks

Neural networks (NNs) are designed based on the structure of the human brain. Many NN architectures have been developed, in which, the feedforward neural network (FFNN) is commonly used in the structural engineering field. In FFNN, the information moves through the network in a one-way direction as illustrated in Fig. 1 [16]. Neurons are organized into multiple layers including the input layer, the hidden layers, and the output layer. The outputs from previous neurons become the inputs of the current neuron after scaling with the corresponding weights. All inputs are summed and then transformed by the activation function.

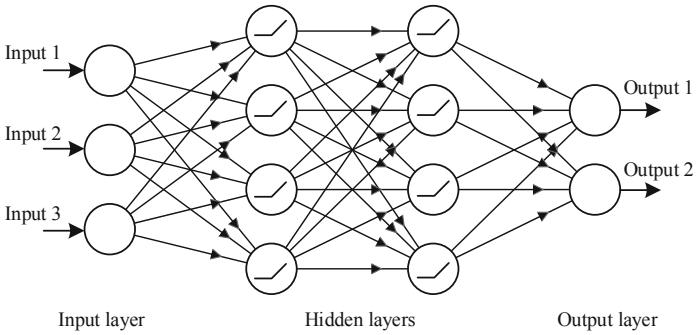

Fig. 1. Typical architecture of an FFNN

It can be expressed as follows. The neuron of j^{th} layer u_j ($j = 1, 2, \ldots, J$) receives a sum of input x_i ($i = 1, 2, \ldots, I$) which is multiplied by the weight w_{ji} and will be transformed into the input of the neuron in the next layer:

$$x_j = f\left(u_j\right) = f\left(\sum_{i=1}^{I} w_{ji} x_i\right) \tag{9}$$

where *f(.)* is the activation function. The most commonly used activation functions are: tanh, sigmoid, softplus, and rectifier linear unit (ReLU). The nonlinear activation function ensures that the network can simulate the complex data.

A NN can has more than one hidden layer, which increases the complexity of the model and can significantly improve prediction accuracy. The loss function is used to measure the error between predicted values and true values. The loss function is chosen depending on the type of task. For regression tasks, the loss functions can be Mean Squared Error (MSE), Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE). To adapts NN for better predictions, the network must be trained. The training process is essentially finding a set of weights in order to minimize the loss function. In other words, it is an optimization process. In the field of machine learning, the commonly used optimization algorithms are stochastic gradient descent (SGD) or Adam optimizer. An effective technique, namely "error backpropagation", developed by Rumelhart et al. [17] is normally used for training.

3 Proposed Surrogate-Based Optimization Approach

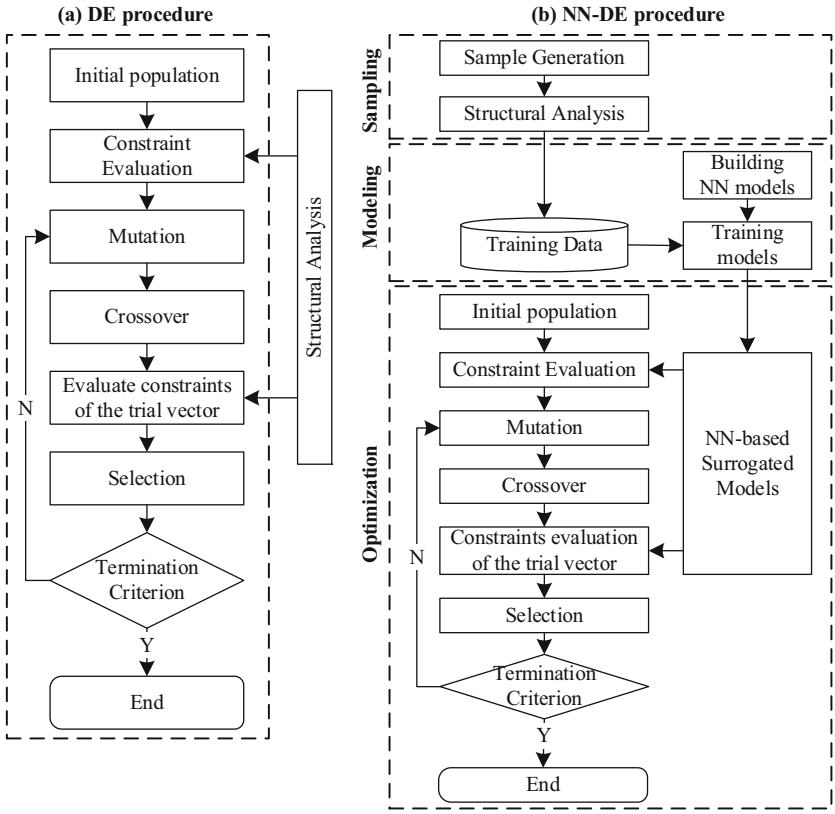

Fig. 2. Optimization workflows

According to the workflow of the conventional DE optimization (Fig. 2(a)), the structural analysis must be performed for every candidate. Consequently, the total number

of structural analyses is too large, leading to time-consuming. To deal with this problem, NNs are used to approximately predict the behavior of the structure instead of FEA. The proposed optimization approach, called NN-DE, consists of three phases: sampling, modeling, and optimization as illustrated in Fig. 2(b).

First of all, some samples are generated. All samples are then analyzed using FEA. The second phase focuses on constructing accurate NN models. After selecting the suitable NN architecture, the models are trained with the datasets obtained from the sampling phase. The performance of the NN models is strongly influenced by the diversity of the training data, which also means depends on the sampling method. Therefore, a method, called Latin Hypercube Sampling, is used to produce the sample points. The NN models that are already trained will be used to evaluate the constraints in the optimization phase.

In this study, both DE and NN-DE approaches are implemented by Python language. The DE algorithm is written based on the code published on R. Storn's website [18]. The structural analysis is conducted with the support of the finite element library PyNiteFEA [19]. NN models are build using the library Keras [20].

4 Computational Experiments

4.1 10-Bar Truss

Design Data. The configuration of the truss are presented in Fig. 3. All members are made of aluminum in which the modulus of elasticity is $E = 10{,}000$ ksi (68,950 MPa) and the density is $\rho = 0.1$ lb/in^3 (2768 kg/m^3). The cross-sectional areas of truss members range from 0.1 to 35 in^2 (0.6 to 228.5 cm^2). Constraints include both stresses and displacements. The stresses of all members must be lower than ± 25 ksi (172.25 Mpa), and the displacements of all nodes are limited to 2 in (50.8 mm).

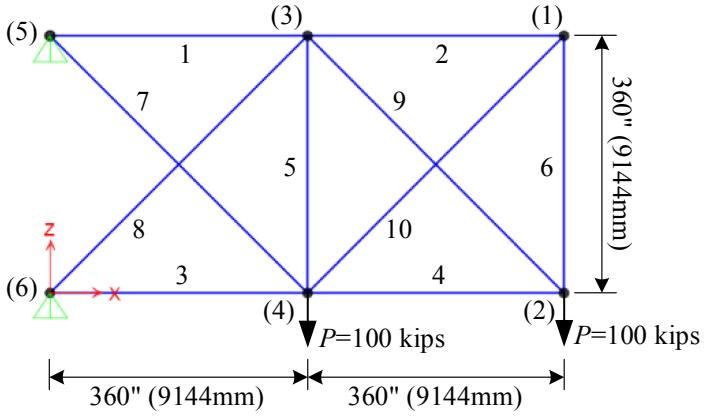

Fig. 3. 10-bar truss layout

Implementation. Firstly, the truss is optimized based on the conventional DE procedure. The design variables are the cross-sectional areas of the truss members. In this problem, each member is assigned to a different section. It means this problem has ten design variables ($D = 10$). The optimization parameters are set after a parameter sensitive analysis as follows: $F = 0.8$; $Cr = 0.7$, $Np = 5 \times D = 50$; maximum iterations $n = 1000$.

For the NN-DE approach, 1000 sample points are initially produced by LHS. Each sample point is a 10-dimensional vector representing the cross-sectional areas of ten corresponding members. Once the input data were generated, the outputs are determined based on FEA. The obtained results are the internal axial forces of members and the displacements of nodes. From the structural engineering point of view, for some sample points, the truss structures are close to being unstable, leading to the large values of the internal forces and the displacements. These sample points are meaningless and they can negatively affect the accuracy of surrogate models. Therefore, the sample points having maximum constraint violation $g(\mathbf{x}) > 1.0$ are removed from the dataset. The final dataset contains about 700 data points.

In the modeling phase, two surrogate models are developed. The first model named 'Model-S' will be used to predict the stresses inside the members while the second model named 'Model-U' will be used to predict the displacements of the free nodes. 'Model-S' has the architecture (10-40-40-40-10) in which ten inputs of the network are the cross-sectional areas of the ten truss members, ten outputs include stresses of ten members and three hidden layers with 40 neurons per layer. 'Model-U' has the architecture (10-20-20-20-4) where four outputs are vertical displacements of four free nodes (1, 2, 3, 4) [21]. ReLU is used as the activation function in this work.

Among available loss functions, MAE is useful if the training data is corrupted with outliers. Therefore, the loss function MAE is used in this work:

$$E_{MAE} = \frac{1}{n} \sum_{i=1}^{n} \left| y_i^{true} - y_i^{pred} \right| \tag{10}$$

where: n is the number of training data points; y_i^{pred} is the predicted value of the network and y_i^{true} is the correct value.

Models are trained for 1000 epochs with a batch size of 10. Before training, the input data should be normalized by dividing to the maximum cross-sectional area (35 in^2). To prevent overfitting, the data is randomly split into the training and the validation subsets with the ratio of 80/20. The loss curves of the two models during the training process are plotted in Fig. 4. The trained models are then employed to predict the stresses and displacements during the optimization process.

The optimization parameters for the NN-DE approach are taken the same as in the conventional DE procedure. The computation is performed on a personal computer with the following configuration: CPU Intel Core i5-5257 2.7 GHz, RAM 8.00 Gb.

Results. Because of the random nature of the search, the optimization is performed for 30 independent runs. The best results of both DE and NN-DE for 30 runs are presented in Table 1 in comparison with other algorithms taken from literature.

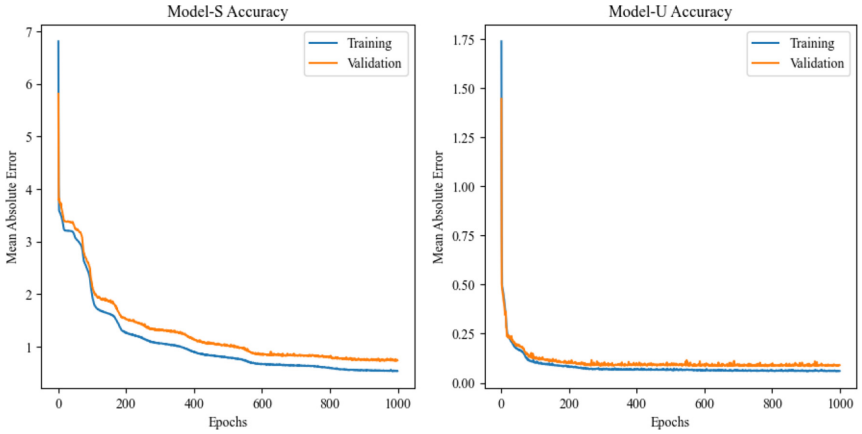

Fig. 4. Loss curves during the training process

Table 1. Comparison of optimum results for 10-bar truss

Member	GA [22]	PSO [22]	ALSSO [22]	DE	NN-DE
1 (in^2)	30.440	33.500	30.4397	30.5603	35.0000
2 (in^2)	0.100	0.100	0.1004	0.1000	0.1000
3 (in^2)	21.790	22.766	23.1599	23.1646	22.7771
4 (in^2)	14.260	14.417	15.2446	15.2025	12.5917
5 (in^2)	0.100	0.100	0.1003	0.1000	0.1000
6 (in^2)	0.451	0.100	0.5455	0.5440	0.1000
7 (in^2)	7.628	7.534	7.4660	7.4632	8.3957
8 (in^2)	21.630	20.392	21.1123	21.0501	22.2952
9 (in^2)	21.360	20.467	21.5191	21.5251	20.7936
10 (in^2)	0.100	0.100	0.1000	0.1001	1.0312
Weight (lb)	4987.00	5024.25	5060.885	5060.8087	5217.7381
Max. constraint violation	1.0140	1.0194	1.0000	1.0000	0.9958

4.2 25-Bar Truss

Design Data. The layout of the truss is presented in Fig. 5. The mechanical properties are the same as in the 10-bar truss. The allowable horizontal displacement in this problem is ± 0.35 in (8.89 mm). Members are grouped into eight groups with the allowable stresses as shown in Table 2. Hence, the number of design variables in this problem $D = 8$. The cross-sectional areas of the members vary between 0.01 in^2 and 3.5 in^2. Two loading cases acting on the structure are given in Table 3.

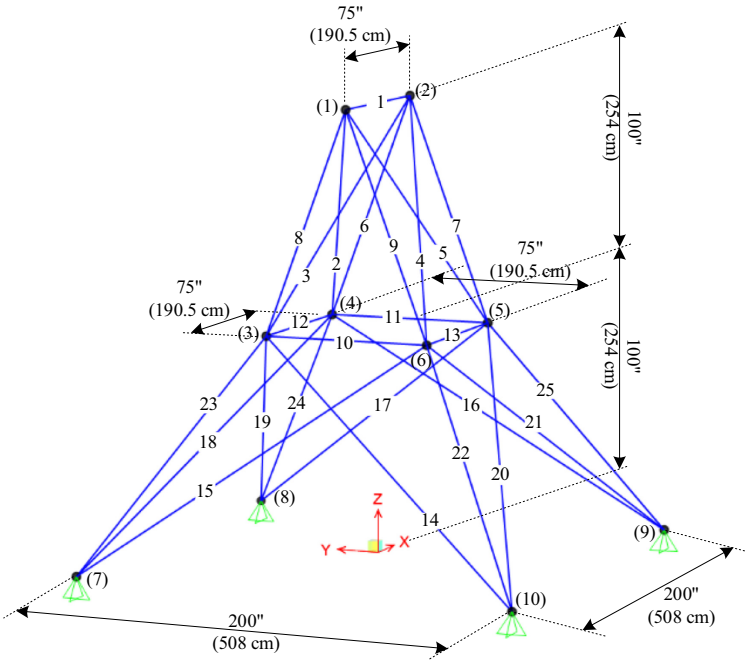

Fig. 5. 25-bar truss layout

Table 2. Allowable stresses for 25-bar truss - unit: ksi (MPa)

Group	Member	Allowable stress for compression	Allowable stress for tension
1	1	35.092 (241.96)	40.0 (275.80)
2	2, 3, 4, 5	11.590 (79.913)	40.0 (275.80)
3	6, 7, 8, 9	17.305 (119.31)	40.0 (275.80)
4	10, 11	35.092 (241.96)	40.0 (275.80)
5	12, 13	35.092 (241.96)	40.0 (275.80)
6	14, 15, 16, 17	6.759 (46.603)	40.0 (275.80)
7	18, 19, 20, 21	6.959 (47.982)	40.0 (275.80)
8	22, 23, 24, 25	11.082 (76.410)	40.0 (275.80)

Implementation. The DE approach and the proposed NN-DE approach are also implemented with the same parameters: $F = 0.8$; $Cr = 0.7$; $Np = 5{\times}D = 40$; $n = 500$.

There are totally 5 NN models which are built in this case. Two models namely 'Model-S-min' and 'Model-S-max' with the architecture (8-50-50-50-25), are used to predict the maximum and the minimum values of stresses inside truss members. Eight inputs of the network are the cross-sectional areas of the eight groups while twenty-five

Table 3. Loading conditions for 25-bar truss - unit: kips (kN)

Node	Case 1			Case 2		
	P_X	P_Y	P_Z	P_X	P_Y	P_Z
1	0.0	20.0 (89)	−5.0 (22.25)	1.0	10.0 (44.5)	−5.0 (22.25)
2	0.0	−20.0 (89)	−5.0 (22.25)	0.0	10.0 (44.5)	−5.0 (22.25)
3	0.0	0.0	0.0	0.5 (2.22)	0.0	0.0
6	0.0	0.0	0.0	0.6 (2.67)	0.0	0.0

outputs are the stress values of 25 members. Three remaining models, namely 'Model-Ux', 'Model-Uy', and 'Model-Uz', have the architecture (8-20-20-20-6) in which six outputs are displacements of six free nodes (1, 2, 3, 4, 5, 6) along three directions (*UX*, *UY*, and *UZ*). Other parameters are completely similar to the 10-bar truss problem. The optimization is also performed for 30 independent runs.

Results. The optimization is also performed for 30 independent runs. The best result of DE, NN-DE, and other algorithms is presented in Table 4.

Table 4. Comparison of optimum results for 25-bar truss

Group	HS [22]	ABC [22]	ALSSO [22]	DE	NN-DE
1 (in^2)	0.047	0.011	0.01001	0.0100	0.0100
2 (in^2)	2.022	1.979	1.983579	1.9173	2.4118
3 (in^2)	2.950	3.003	2.998787	2.9793	3.2004
4 (in^2)	0.010	0.01	0.010008	0.0100	0.0100
5 (in^2)	0.014	0.01	0.010005	0.0100	0.0100
6 (in^2)	0.688	0.69	0.683045	0.6801	0.4485
7 (in^2)	1.657	1.679	1.677394	1.7217	1.6081
8 (in^2)	2.663	2.652	2.66077	2.6601	2.5083
Weight (lb)	544.38	545.193	545.1057	543.7736	545.9119
Max. constraint violation	1.0238	1.0212	1.0218	1.0000	1.0697

4.3 Performance Comparisons

The performances of the DE approach and the NN-DE approach are given in Table 5.

Based on the obtained results, some observations can be summarized as follows.

First of all, it can be noted that the DE algorithm is capable of finding the global optimum. The conventional DE optimization achieves the same results in comparison with

Table 5. Comparison of computation time

Problem	Approach	Number of FEA	Computation time (s)			
			FEA	Training	Optimization	Total
10-bar truss	DE	50,050	1,757	–	–	1,757
	NN-DE	1,000	36	234	54	324
25-bar truss	DE	20,040	4,426	–	–	4,426
	NN-DE	1,000	232	279	64	575

other algorithm like GA, PSO, HS, ALSSO, etc. Minor differences in results obtained from different algorithms due to the use of different FEA codes.

Secondly, the computation times of the NN-DE approach are very small compared to the conventional DE optimization. For the 10-bar truss, the computation times of the DE and NN-DE approaches are 1757 s and 324 s, respectively. These values are 4426 s and 575 s for the 25-bar truss. It means the NN-DE approach is more than 5 times faster than the conventional DE approach. It is achieved by a significant reduction in the number of FEAs.

Furthermore, the errors of the optimum weight between DE and NN-DE for two problems are 3.1% and 0.4%, respectively. The reason for the errors is the models' inaccuracy in predicting structural behavior. However, the coefficient of determination (R^2) between the cross-sectional areas found by DE and NN-DE for 10-bar truss and 25-bar truss are 0.981 and 0.971, respectively. It indicates a good similarity and linear correlation between the results of the DE and NN-DE approaches.

5 Conclusions

In this paper, a hybrid approach, called NN-DE, which is combined Neural Networks and Differential Evolution algorithm is proposed. By using Neural Networks as surrogated models instead of FEA, the computation times of the evolutionary optimization is significantly reduced. Thus, the proposed approach has a high potential for large-scale structures. Besides, the optimum results of the proposed approach have still slightly errors compared to the conventional procedure. Therefore, it is important to notice that the results obtained from the proposed approach are not exact, but it can be considered as a good suggestion in the practical design. In the future, the methods to improve the accuracy of the Neural Network models should be studied.

Acknowledgment. This work was supported by the Domestic Ph.D. Scholarship Programme of Vingroup Innovation Foundation.

References

1. Rajeev, S., Krishnamoorthy, C.S.: Discrete optimization of structures using genetic algorithms. J. Struct. Eng. **118**(5), 1233–1250 (1992)

2. Beyer, H.G.: The Theory of Evolution Strategies. Springer, Heidelberg (2001). https://doi.org/10.1007/978-3-662-04378-3s
3. Price, K.V., Storn, R.M., Lampien, J.A.: Differential Evolution: A Practical Approach to Global Optimization. Springer, Heidelberg (2005). https://doi.org/10.1007/3-540-31306-0
4. Eberhart, R., Kennedy, J.: Particle swarm optimization. In: Proceedings of ICNN 1995-International Conference on Neural Networks, vol. 4, pp. 1942–1948. IEEE (1995)
5. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
6. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Technical report - TR06, vol. 200, pp. 1–10 (2005)
7. Latif, M.A., Saka, M.P.: Optimum design of tied-arch bridges under code requirements using enhanced artificial bee colony algorithm. Adv. Eng. Softw. **135**, 102685 (2019)
8. Papadrakakis, M., Lagaros, N.D., Tsompanakis, Y.: Optimization of large-scale 3-D trusses using evolution strategies and neural networks. Int. J. Space Struct. **14**(3), 211–223 (1999)
9. Kaveh, A., Gholipour, Y., Rahami, H.: Optimal design of transmission towers using genetic algorithm and neural networks. Int. J. Space Struct. **23**(1), 1–19 (2008)
10. Krempser, E., Bernardino, H.S., Barbosa, H.J., Lemonge, A.C.: Differential evolution assisted by surrogate models for structural optimization problems. In: Proceedings of the international conference on computational structures technology (CST), vol. 49. Civil-Comp Press (2012)
11. Penadés-Plà, V., García-Segura, T., Yepes, V.: Accelerated optimization method for low-embodied energy concrete box-girder bridge design. Eng. Struct. **179**, 556–565 (2019)
12. Vesterstrom, J., Thomsen, R.: A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In: Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753), Portland, USA, vol. 2, pp. 1980–1987. IEEE (2004)
13. Hieu, N.T., Tuan, V.A.: A comparative study of machine learning algorithms in predicting the behavior of truss structures. In: Proceeding of the 5th International Conference on Research in Intelligent and Computing in Engineering RICE 2020. Springer (2020). (accepted for publication)
14. Storn, R., Price, K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**(4), 341–359 (1997)
15. Lampinen, J.: A constraint handling approach for the differential evolution algorithm. In: Proceedings of the 2002 Congress on Evolutionary Computation (IEEE Cat. No. 02TH8600), Portland, USA, vol. 2, pp. 1468–1473. IEEE (2002)
16. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
17. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**(6088), 533–536 (1986)
18. Differential Evolution Code Homepage. https://www.icsi.berkeley.edu/~storn/code.html. Accessed 28 Jan 2020
19. PyNiteFEA Homepage. https://pypi.org/project/PyNiteFEA/. Accessed 22 Apr 2020
20. Keras Document Homepage. https://keras.io/. Accessed 22 Apr 2020
21. Lee, S., Ha, J., Zokhirova, M., Moon, H., Lee, J.: Background information of deep learning for structural engineering. Arch. Comput. Meth. Eng. **25**(1), 121–129 (2018)
22. Du, F., Dong, Q.Y., Li, H.S.: Truss structure optimization with subset simulation and augmented Lagrangian multiplier method. Algorithms **10**(4), 128 (2017)