Self-Directed Online Machine Learning for Topology Optimization

Changyu Deng
University of Michigan-Ann Arbor https://orcid.org/0000-0001-8339-4014
Yizhou Wang
Northeastern University https://orcid.org/0000-0003-1601-9649
Can Qin
Northeastern University
Wei Lu (weilu@umich.edu)
University of Michigan–Ann Arbor https://orcid.org/0000-0002-4851-1032

Article

Keywords: Deep Neural Network, Finite Element Method, Compliance Minimization Problems, Fluid-structure Optimization Problems, Heuristic Methods, Multi-dimensional Optimization Problems

Posted Date: January 25th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-146826/v1

Self-Directed Online Machine Learning for Topology Optimization

Changyu Deng[1], Yizhou Wang[2], Can Qin[2], and Wei Lu[1,3,*]

[1]Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109, United States
[2]Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115, United States
[3]Department of Materials Science and Engineering, University of Michigan, Ann Arbor, MI 48109, United States
[*]Corresponding author: weilu@umich.edu

Abstract

Topology optimization by optimally distributing materials in a given domain requires gradient-free optimizers to solve highly complicated problems. However, with hundreds of design variables or more involved, solving such problems would require millions of Finite Element Method (FEM) calculations whose computational cost is huge and impractical. Here we report a Self-directed Online Learning Optimization (SOLO) which integrates Deep Neural Network (DNN) with FEM calculations. A DNN learns and substitutes the objective as a function of design variables. A small number of training data is generated dynamically based on the DNN's prediction of the global optimum. The DNN adapts to the new training data and gives better prediction in the region of interest until convergence. Our algorithm was tested by compliance minimization problems and fluid-structure optimization problems. It reduced the computational time by 2 $\sim$ 5 orders of magnitude compared with directly using heuristic methods, and outperformed all state-of-the-art algorithms tested in our experiments. This approach enables solving large multi-dimensional optimization problems.

Main

Distributing materials in a domain to optimize performance is a significant topic in many fields, such as solid mechanics, heat transfer, acoustics, fluid mechanics, materials design and various multiphysics disciplines[1]. Many numerical approaches[2] have been developed since 1988, where the problems are formulated by density, level set, phase field, topological derivative or other methods[3]. Typically, these approaches require gradient-based optimizers, such as the Method of Moving Asymptotes (MMA), and thus have various restrictions on the properties of governing equations and optimization constraints to allow for fast computation of gradients. Because of the intrinsic limitation of gradient-based algorithms, the majority of existing approaches have only been applied to simple problems, since they would fail as soon as the problem becomes complicated such as involving varying signs on gradients or non-linear constraints[4]. To address these difficulties, gradient-free methods have been developed which play a significant role in

overcoming the tendency to be trapped in a local minimum.

Several researchers have attempted to implement gradient-free optimizers, all of which are stochastic and heuristic methods. For instance, Hajela et al. applied a Genetic Algorithm (GA) to a truss structure optimization problem to reduce weight[5]. Shim and Manoochehri minimized the material use subject to maximum stress constraints by a Simulated Annealing (SA) approach[6]. Besides these two popular methods, other algorithms have been investigated as well, such as ant colonies[7,8], particle swarms[9], harmony search[10], and bacterial foraging[11]. Gradient-free methods have four advantages over gradient-based methods[12]: better optima, applicable to discrete designs, free of gradients and efficient to parallelize. However, the major disadvantage of the methods is their high computational cost from calling the objective functions, which becomes prohibitively expensive for large systems[3].

Machine learning has been used in sequential model-based optimization (SMBO) targeting at expensive objective function evaluation[13,14]. For instance, Bayesian optimization (BO)[15] uses a Gaussian prior to approximate the conditional probability distribution of an objective $p(y|x)$ where $y = F(x)$ is the objective and x is the design variable (vector); then the unknown regions can be estimated by the probability model. In Covariance Matrix Adaptation Evolution Strategy (CMA-ES)[16], a multivariable Gaussian distribution is used to sample new queries. However, these methods are not designed for large-scale and high-dimensional problems. Despite some improvement to scale up these algorithms[17,18], none of them has been implemented in topology optimization to the best of our knowledge.

There are some reports on leveraging machine learning to reduce the computational cost of topology optimization[19–25]. Generative models are used to predict solutions of the same problem under different conditions, after being trained by optimized solutions from gradient-based methods. For example, Yu et al.[26] used 100,000 optimal solutions to a simple compliance problem with various boundary forces and the optimal mass fractions to train a neural network consisting of Convolutional Neural Network (CNN) and conditional Generative Adversarial Network (cGAN), which can predict near-optimal designs of mass fraction for any given boundary forces. However, these schemes are not topology optimization algorithms: they rely on existing optimal designs as the training data. The predictions are restricted by the coverage of the training datasets. To consider different domain geometry or constraints, new datasets and networks would be required. Besides, the designs predicted by the networks are close to, but still different from the optimal designs.

Here we propose a gradient-free algorithm called Self-directed Online Learning Optimization (SOLO). A DNN is used to map designs to objectives as a surrogate model to approximate and replace the original function which is expensive to calculate. A heuristic optimization algorithm finds the possible optimal design according to DNN's prediction. Based on the optimum, new query points are dynamically generated and evaluated by the Finite Element Method (FEM) to serve as additional training data. The loop of such self-directed online learning is repeated until convergence. This iterative learning scheme, which can be categorized as an SMBO algorithm, takes advantage of the searching abilities of heuristic methods and the high computational speed of DNN. Theoretical convergence rate is derived under some assumptions. To show its performance, we test the algorithm by two compliance minimization problems (designing solid so that

the structure achieves maximum stiffness for given loading) and two fluid-structure optimization problems (designing fluid tunnel to minimize fluid pressure loss for given inlet speed). Our algorithm reduces the computational cost by at least two orders of magnitude compared with directly applying heuristic methods. In addition to benchmarks from gradient-based solvers, we compare our algorithm with an offline version (where all training data are randomly generated), Generalized Simulated Annealing (GSA), BO, CMA-ES and a recent algorithm based on reinforcement learning[27].

Problem formulation and algorithm description

Consider the following topology optimization problem: in a design domain Ω, find the material distribution $\rho(\mathbf{x})$ that could take either 0 (void) or 1 (solid) at point $\mathbf{x}$ to minimize the objective function F, subject to a volume constraint $G_0 \leq 0$ and possibly M other constraints $G_j \leq 0 (j = 1, ..., M)$. Mathematically, this problem can be written as[4]

$$\begin{aligned} &\min_{\rho=\rho(\mathbf{x})} F(\rho) \\ &\begin{cases} G_0(\rho) = \int_\Omega \rho(\mathbf{x})\, \mathrm{d}V - V_0 \leq 0 \\ G_j(\rho) \leq 0, \quad j = 1, ..., M \\ \rho(\mathbf{x}) = 0 \text{ or } 1, \quad \forall \mathbf{x} \in \Omega \end{cases}, \end{aligned} \tag{1}$$

where V_0 denotes the given volume. To solve such a problem numerically, the domain Ω is discretized into finite elements to describe the density distribution by N nodal or elemental values,

$$\begin{aligned} &\min_{\rho=(\rho_1,\rho_2,...,\rho_N)} F(\rho_1, \rho_2, ..., \rho_N) \\ &\begin{cases} G_0(\rho) = \sum_{i=1}^{N} w_i \rho_i - V_0 \leq 0 \\ G_j(\rho) \leq 0, \quad j = 1, ..., M \\ \rho(\mathbf{x}) = 0 \text{ or } 1, \quad \forall \mathbf{x} \in \Omega \end{cases}, \end{aligned} \tag{2}$$

where w_i denotes the weight of integration. In gradient-based methods, ρ_i is assumed to be continuous from 0 to 1. Thus, the problem is formulated as

$$\begin{aligned} &\min_{\rho=(\rho_1,\rho_2,...,\rho_N)} F(\rho_1, \rho_2, ..., \rho_N) \\ &\begin{cases} G_0(\rho) = \sum_{i=1}^{N} w_i \rho_i - V_0 \leq 0 \\ G_j(\rho) \leq 0, \quad j = 1, ..., M \\ 0 \leq \rho_i \leq 1, \quad i = 1, ..., N \end{cases}. \end{aligned} \tag{3}$$

Our algorithm can be applied to both binary (Eq.(2)) and continuous (Eq.(3)) variables. In this section, we discuss the latter since it is more general.

In many applications, the objective function is quite complicated and time-consuming to calculate, since it requires solving partial differential equations by, for instance, FEM. To reduce the

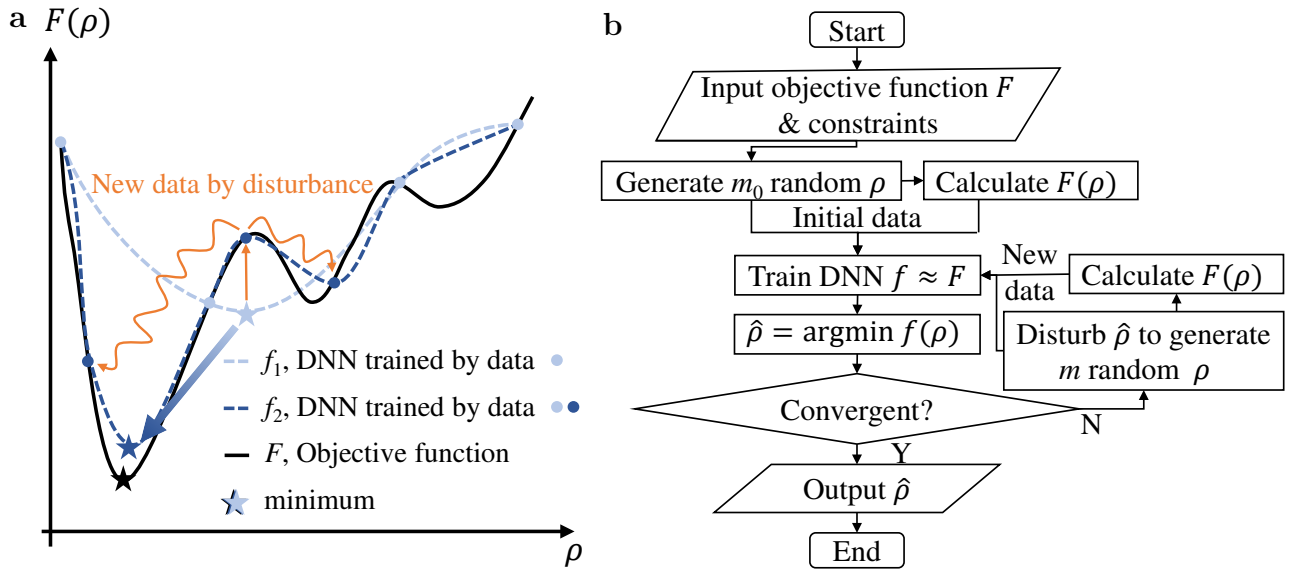

Fig. 1: Schematics of the proposed self-directed online learning optimization. **a**, Schematic illustration of self-directed online training. The initial batch of training data (light blue dots) is randomly located. The DNN f_1 (dashed light-blue line) trained on first batch of data only gives a rough representation of the true objective function F (solid black line). The second batch training data (dark blue dots) are generated by adding disturbance (orange curve) to the minimum of f_1. After trained with two batches, the DNN f_2 (dashed dark-blue line) is more refined around the minimum (the region of interest), while remains almost the same at other locations such as the right convex part. f_2 is very close to finding the exact global minimum point. **b**, Flow diagram of the algorithm.

number of FEM calculations and accelerate gradient-free optimization, we build a DNN to evaluate the objective function. In a naive way, the entire domain of the objective function should be explored to generate the training data. This would incur a huge number of FEM calculations. However, we only care about the function values close to the global optimum and do not require precise predictions in irrelevant regions. In other words, most information about the objective function in the domain is unnecessary except the details around the optimum. So we do not need to generate data to train those irrelevant regions.

An intuitive explanation is shown in Fig. 1a. In a 1D minimization example, we can generate a small dataset to train the DNN and refine the mesh around the minimum obtained from the current prediction to achieve higher resolution at the place of interest in the next iteration. After several batches, the minimum of the predicted function would converge to that of the objective function.

Fig. 1b shows the flow diagram of the proposed algorithm. A small batch of random density arrays ρ satisfying the constraints in Eq.(3) is generated as the training data and inputted into the DNN, together with corresponding objective function values $F(\rho)$ calculated by FEM. At this stage, the DNN has a certain level of ability to predict the function values based on the density arrays. Next, the global minimum of the objective function $f(\rho)$ is calculated by a

heuristic algorithm (we use a small letter f to denote the DNN-approximated function). After obtaining the optimized array $\widehat{\rho}$, more training data are generated accordingly. Inspired by the concept of GA[28], the disturbance we add to the array is more than a small perturbation, and is categorized as mutation and crossover. Mutation means replacing one or several design variables with random numbers, while crossover means exchanging several values in the array. Then constraints are checked and enforced. The self-directed learning and optimization process stops when the value of the objective function $F(\widehat{\rho})$ does not change anymore or the computation budget is exhausted.

This algorithm can converge provably under some mild assumptions. Given the total number of training data n_{train}, for any trained DNN with small training error, we have

$$(F(\widehat{\rho}) - F^*)^2 \leq \widetilde{O}\Big(\frac{C}{\sqrt{n_{train}}}\Big), \tag{4}$$

where C is a constant related to some inherent properties of F and DNN, F^* is the global minimum of F, and $\widetilde{O}$ omits log terms. This result states that when our trained DNN can fit the training data well, then our algorithm can converge to the global optimal value. We provide convergence guarantee with concrete convergence rate for our proposed algorithm, and to the best of our knowledge, this is the first non-asymptotic convergence result for heuristic optimization methods using DNN as a surrogate model. The detailed theory and its derivation are elaborated in Supplementary Section 2.

Examples and results

In this section, we will apply the algorithm to four classic examples (covering both continuous and binary variables): two compliance minimization problems and two fluid-structure optimization problems.

Compliance minimization. We first test the algorithm on two simple continuous compliance minimization problems. We show that our algorithm can converge to global optimum and is faster than other gradient-free methods.

As shown in Fig. 2a, a square domain is divided evenly by a 4×4 mesh. A force downward is applied at the top right edge; the bottom left edge is set as a roller (no vertical displacement); the right boundary is set to be symmetric. There are 25 nodal design variables to control the material distribution, i.e. density ρ. Our goal is to find the density $\rho_i (i = 1, 2, ..., 25)$, subject to a volume constraint of 0.5, such that the elastic energy E of the structure is minimized, equivalent to minimizing compliance or the vertical displacement where the external force is applied. Formally,

$$\min_{\rho \in [0,1]^N} \widetilde{E}(\rho) = E(\rho)/E(\rho_O), \tag{5}$$

where $\rho_O = (0.5, 0.5, ..., 0.5)$. The constraint is

$$w \cdot \rho \leq 0.5, \tag{6}$$

where w denotes the vector of linear Gaussian quadrature. In Eq.(5) we use the dimensionless elastic energy $\widetilde{E}(\rho)$, defined as the ratio of elastic energy of the structure with optimized material

distribution to that of the reference uniform distribution (the material density is 0.5 everywhere in the domain). The elastic energy is calculated by FEM from the Young's modulus in the domain, which is related to density by the popular Simplified Isotropic Material with Penalization (SIMP) method,[29]

$$Y(\rho(\mathbf{x})) = Y_0\rho(\mathbf{x})^3 + \varepsilon\left[1-\rho(\mathbf{x})^3\right], \tag{7}$$

where Y and Y_0 denote the Young's moduli as a variable and a constant respectively, ε is a small number to avoid numerical singularity and $\rho(\mathbf{x})$ is the material density at a given location $\mathbf{x}$ interpolated linearly by the nodal values of the element.

For benchmark, we use a traditional gradient-based algorithm, the Method of Moving Asymptotes (MMA), to find the optimized solution (Fig. 2d).

For our proposed method, we use 100 random arrays to initialize the DNN. Then Generalized Simulated Annealing (GSA) is used to obtain the minimum $\widehat{\rho}$ based on the DNN's prediction. Afterwards, 100 additional samples will be generated by adding disturbance to $\widehat{\rho}$. Such a loop continues until convergence.

We compare our proposed method, Self-directed Online Learning Optimization (SOLO), with four other algorithms. In Fig. 2b, SOLO converges at $n_{train} = 501$. "Offline" denotes a naive implementation to couple DNN with GSA, which trains a DNN offline by n_{train} random samples and then uses GSA to search for the optimum, without updating the DNN. As expected, the elastic energy decreases with the number of accumulated training samples n_{train}. This is because more training data will make the DNN estimate the elastic energy more accurately. Yet it converges much slower than SOLO and does not work well even with $n_{train} = 2,000$. More results are shown in Supplementary Fig. 1. SS denotes Stochastic Search, which uses current minimum (the minimum of existing samples) to generate new searching samples; the setup is the same as SOLO except that the base design $\widehat{\rho}$ is obtained from the current minimum instead of a DNN. Comparing SS with SOLO, we can conclude that the DNN in SOLO gives a better searching direction than using existing optima. CMA-ES denotes Covariance Matrix Adaptation Evolution Strategy with multi-variable Gaussian prior. BO denotes Bayesian Optimization with Gaussian distribution as the prior and expected improvement as the acquisition function. Our method outperforms all these methods in terms of convergence speed. CMA-ES ranks the second with a solution similar to (3% higher objective) SOLO at $n_{train} = 2,000$.

To assess inference accuracy in online and offline learning, we compare the DNN-predicted energy with that calculated by FEM on the same material distribution. The relative error is defined by $[e(\widehat{\rho}) - E(\widehat{\rho})]/E(\widehat{\rho})$ where $e(\widehat{\rho})$ and $E(\widehat{\rho})$ denote energy calculated by DNN and FEM respectively. The energy prediction error is shown in Fig. 2c. When n_{train} is small, both networks overestimate the energy since their training datasets, composed of randomly distributed density values, correspond to higher energy. As n_{train} increases, the error of SOLO fluctuates around zero since solutions with low energy are fed back to the network.

The solution of SOLO using 501 samples is presented in Fig. 2e, whose energy is 0.298, almost the same as that of the benchmark in Fig. 2d. With higher n_{train}, the solution from SOLO becomes closer to that of the benchmark (the evolution of optimized structures is shown in Supplementary Fig. 2). In Fig. 2f, the energy is the same as the benchmark. The material distribution in Fig. 2f

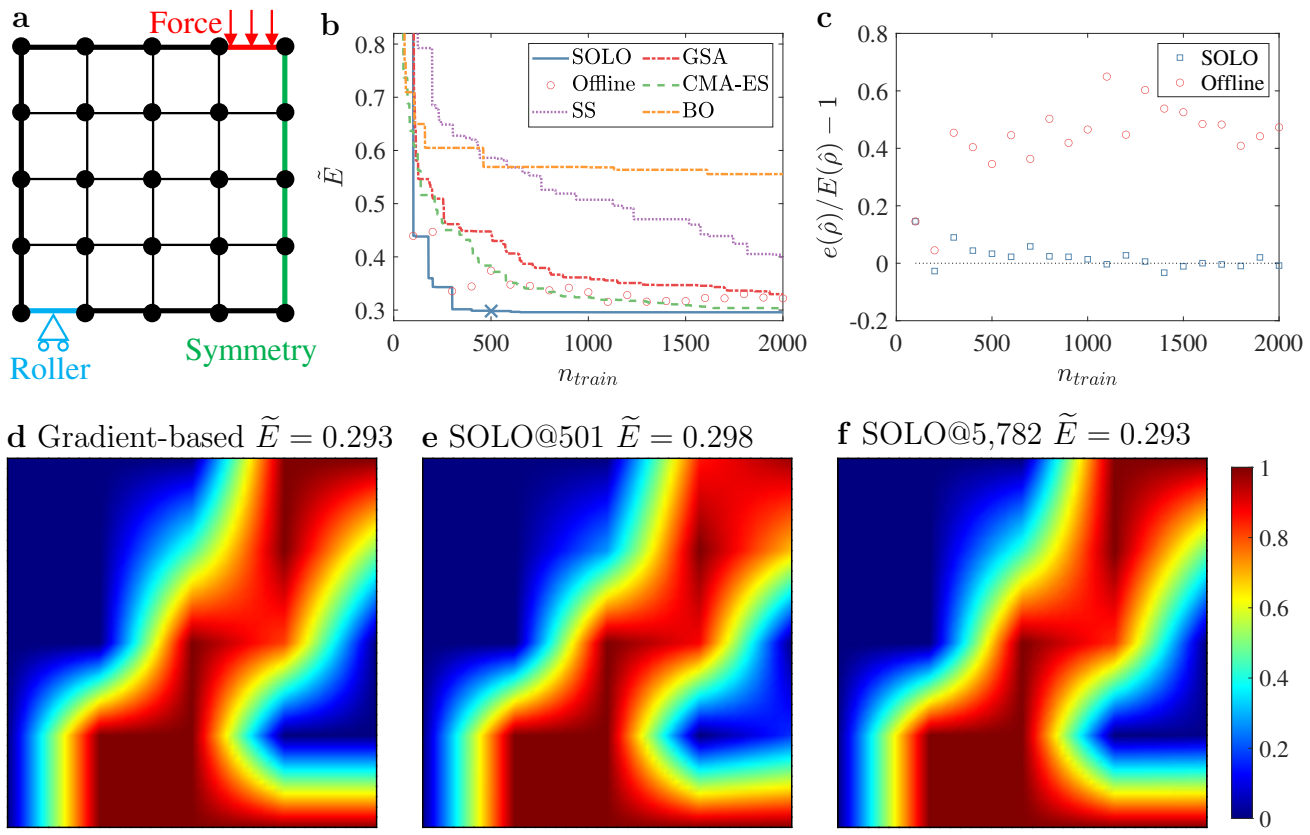

Fig. 2: Setup and results of a compliance minimization problem with 5×5 design variables. **a**, Problem setup. **b**, Best dimensionless energy with a total of n_{train} accumulated training samples. SOLO denotes our proposed method where the cross "X" denotes the convergence point (presented in **e**), "Offline" denotes training a DNN offline and then uses GSA to search for the optimum without updating the DNN, SS denotes Stochastic Search, which is the same as SOLO except that $\widehat{\rho}$ in each loop is obtained by the minimum of existing samples, CMA-ES denotes Covariance Matrix Adaptation Evolution Strategy, BO denotes Bayesian Optimization. SOLO converges the fastest among these methods. **c**, Energy prediction error of $\widehat{\rho}$ relative to FEM calculation of the same material distribution. **d**, Optimized design produced by the gradient-based method. $\widetilde{E}=0.293$. **e**, Optimized design produced by SOLO. $n_{train}=501$ and $\widetilde{E}=0.298$. **f**, Optimized design produced by SOLO. $n_{train}=5,782$ and $\widetilde{E}=0.293$. In **d-f**, dark red denotes $\rho=1$ and dark blue denotes $\rho=0$, as indicated by the right color scale bar.

does not differ much from that in Fig. 2e. In fact, using only 501 samples is sufficient for the online training to find the optimized material distribution. We find that in our problem and optimization setting, the GSA needs about 2×10^5 function evaluations to obtain the minimum of DNN. Since the DNN approximates the objective function, we estimate GSA needs the same number of evaluations when applying to the objective, then it means 2×10^5 FEM calculations are required if directly using GSA. From this perspective, SOLO reduces the number of FEM calculations to 1/400.

A similar problem with a finer mesh having 121 (11×11) design variables is shown in Fig. 3a.

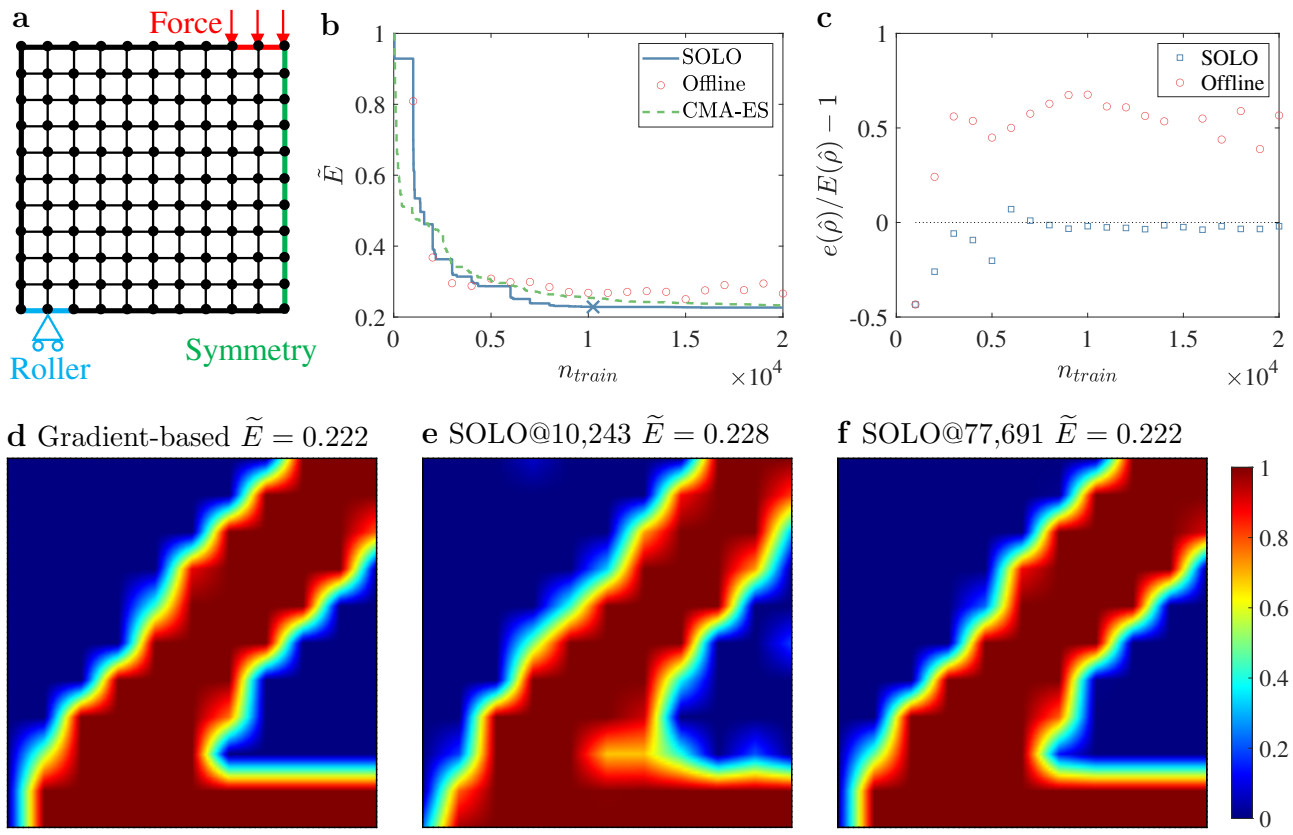

Fig. 3: Setup and results of a compliance minimization problem with 11×11 design variables. **a**, Problem setup. **b**, Best dimensionless energy with a total of n_{train} accumulated training samples. SOLO denotes our proposed method where the cross "X" denotes the convergence point (presented in **e**), "Offline" denotes training a DNN offline and then uses GSA to search for the optimum without updating the DNN, CMA-ES denotes Covariance Matrix Adaptation Evolution Strategy. SOLO converges the fastest among these methods. **c**, Energy prediction error of $\widehat{\rho}$ relative to FEM calculation of the same material distribution. **d**, Optimized design produced by the gradient-based method. $\widetilde{E}=0.222$. **e**, Optimized design produced by SOLO. $n_{train}=10,243$ and $\widetilde{E}=0.228$. **f**, Optimized design produced by SOLO. $n_{train}=77,691$ and $\widetilde{E}=0.222$. In **d-f**, dark red denotes $\rho=1$ and dark blue denotes $\rho=0$, as indicated by the right bar.

The benchmark solution from MMA is shown in Fig. 3d, whose energy is 0.222. The trends in Fig. 3b and c are similar to those in Fig. 2 with a coarse mesh. Fig. 3b shows that SOLO converges at $n_{train}=10,243$, giving $\widetilde{E}=0.228$. Our method outperforms CMA-ES, the best algorithm according to Fig. 2b. The material distribution solutions are shown in Fig. 3e and f. The configuration of SOLO is the same as that for the coarse mesh except that each loop has 1,000 incremental samples and GSA performs 4×10^6 function evaluations. Compared with directly using GSA, SOLO reduces the number of FEM calculations to 1/400 as well. The evolution of optimized structures is shown in Supplementary Fig. 3.

Fluid-structure optimization. In the following two problems, we leverage our algorithm to

address discrete fluid-structure optimization. We want to show that our method outperforms the gradient-based method and a recent algorithm based on reinforcement learning[27].

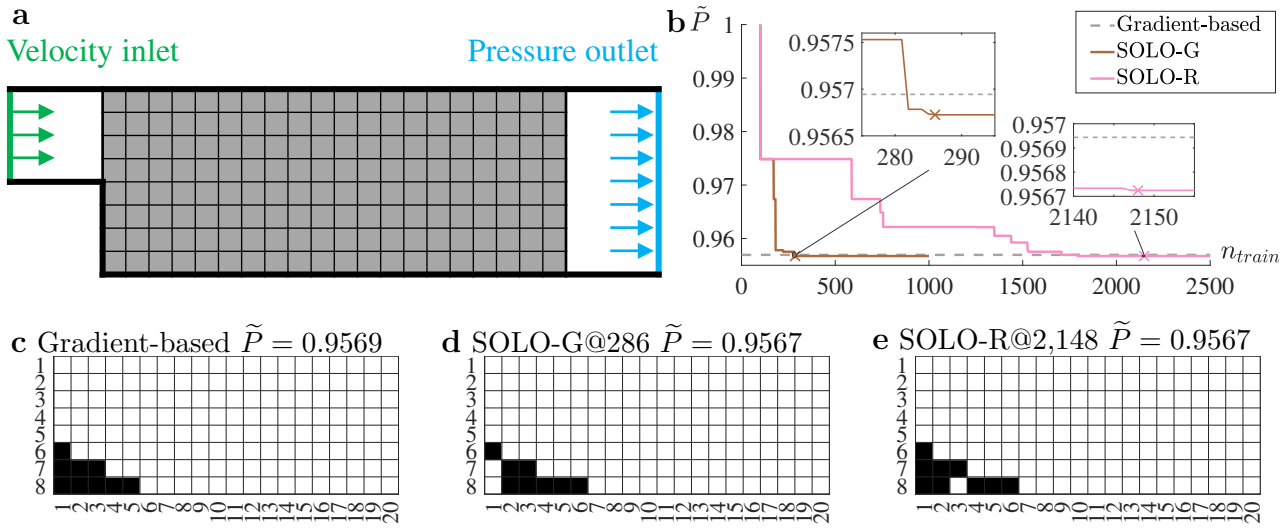

Fig. 4: Setup and results of a fluid-structure optimization problem with 20×8 design variables. **a**, Problem setup. The vertical green line denotes the inlet while the vertical blue line denotes the outlet. **b**, Dimensionless inlet pressure versus n_{train}, the number of accumulated training samples. SOLO-G denotes a greedy version of our proposed method, SOLO-R denotes the regular version of our proposed method. The horizontal dashed line denotes the solution from the gradient-based method. The cross "X" denotes the convergence point (presented in **d** and **e**, respectively). **c**, Optimized design obtained by the gradient-based method. $\widetilde{P} = 0.9569$. **d**, Optimized design obtained by SOLO-G. $n_{train} = 286$ and $\widetilde{P} = 0.9567$. **e**, Optimized design obtained by SOLO-R. $n_{train} = 2,148$ and $\widetilde{P} = 0.9567$. In **c**-**e**, black denotes $\rho = 1$ (solid) and white denotes $\rho = 0$ (void). These solutions are equivalent since the flow is blocked by the black squares forming the ramp surface and the white squares within the ramp at the left bottom corner are irrelevant.

As shown in Fig. 4a, the fluid enters the left inlet at a given velocity perpendicular to the inlet, and flows through the channel bounded by walls to the outlet where the pressure is set as zero. In the 20×8 mesh, we add solid blocks to change the flow field to minimize the friction loss when the fluid flows through the channel. Namely, we want to minimize the normalized inlet pressure

$$\min_{\rho \in \{0,1\}^N} \widetilde{P}(\rho) = P(\rho)/P(\rho_O), \tag{8}$$

where P denotes the average inlet pressure and $\rho_O = (0, 0, ..., 0)$ indicates no solid in the domain. As for the fluid properties, we select a configuration with a low Reynolds number for stable steady solution[30], specifically,

$$\mathrm{Re} = \frac{DvL}{\mu} = 40, \tag{9}$$

where D denotes fluid density, μ denotes viscosity, v denotes inlet velocity and L denotes inlet width (green line).

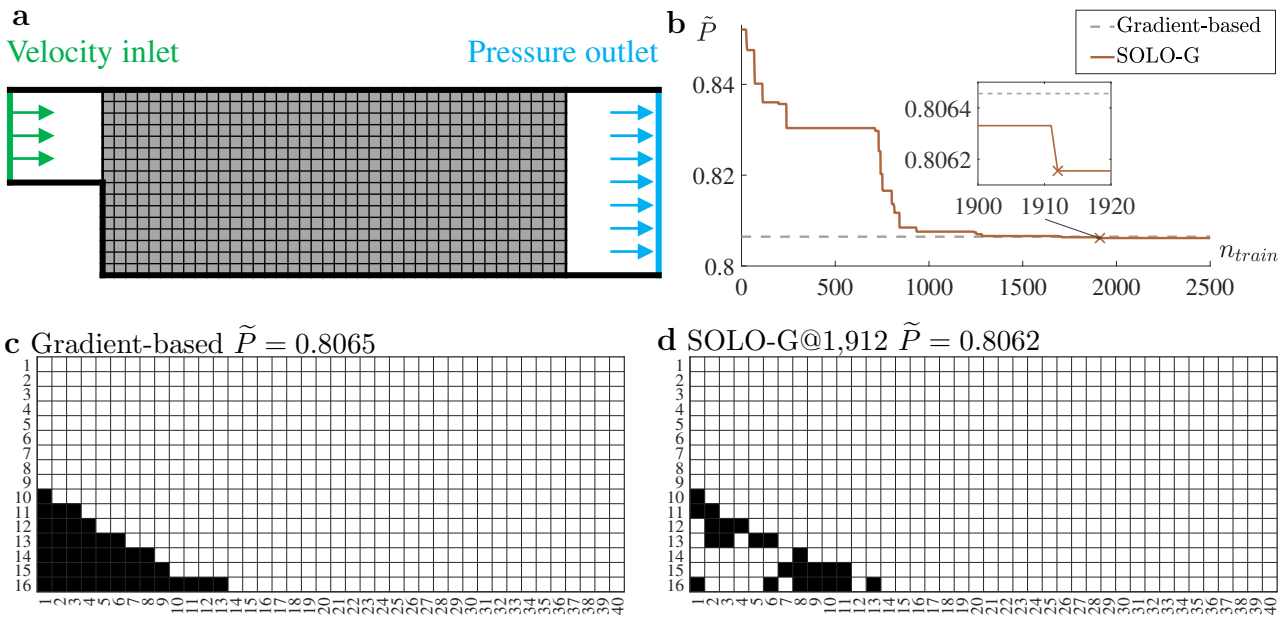

Fig. 5: Setup and results of a fluid-structure optimization problem with 40×16 design variables. a, Problem setup. **b**, Dimensionless inlet pressure versus n_{train}, the number of accumulated training samples. SOLO-G denotes a greedy version of our proposed method, where the cross "X" denotes the convergence point (presented in **d**). The horizontal dashed line denotes the solution from the gradient-based method. **c**, Optimized design obtained by the gradient-based method. $\widetilde{P} = 0.8065$. **d**, Optimized design obtained by SOLO-G. $n_{train} = 1,912$ and $\widetilde{P} = 0.8062$. In **c**,**d**, black denotes $\rho = 1$ (solid) and white denotes $\rho = 0$ (void). The SOLO-G result in **d** has two gaps at the 7th and 12th columns, while the gradient-based result in **c** gives a smooth ramp. We try filling the gaps and find that their existence indeed reduces pressure, which demonstrates the powerfulness of our gradient-free method.

For the benchmark, we use a typical gradient-based algorithm which adds an impermeable medium to change binary variables to continuous ones[31]. It uses the adjoint method to derive gradients and MMA as the solver. The solution is presented in Fig. 4c. The solid blocks form a ramp at the left bottom corner for a smooth flow expansion.

We use two variants of our algorithm. One is denoted as SOLO-G, a greedy version of SOLO where additional 10 samples produced in each loop are all from the DNN's prediction. The initial batch is composed of a solution filled with zeros and 160 solutions each of which has a single element equal to one and others equal to zero. The pressure values corresponding to these designs are calculated by FEM. These 161 samples are used to train a DNN. Next, Binary Bat Algorithm (BBA) is used to find the minimum of the DNN. The top 10 solutions (after removing repeated ones) encountered during BBA searching will be used as the next batch of training data. The other variant, denoted as SOLO-R, is a regular version of SOLO where each loop has 100 incremental samples. 10 of them are produced in the same way as SOLO-G whereas the rest 90 are generated by adding disturbance to the best solution predicted by the DNN. Similar to the compliance minimization problems, the disturbance includes mutation and crossover.

As shown in Fig. 4b, SOLO-G and SOLO-R converge to the same objective function value $\widetilde{P} = 0.9567$ at $n_{train} = 286$ and $n_{train} = 2,148$ respectively. Their solutions are equivalent, shown in Fig. 4d and e. Intermediate solutions from SOLO-G are shown in Supplementary Fig. 4. We obtain the optimum better than the gradient-based method ($\widetilde{P} = 0.9569$) after only 286 FEM calculations. For comparison, a recent topology optimization work based on reinforcement learning used the same geometry setup and obtained the same solution as the gradient-based method after thousands of iterations[27]; our approach demonstrates better performance. Compared with directly using BBA which requires 10^8 evaluations, SOLO-G reduces FEM calculations to by orders of magnitude to about $1/(3 \times 10^5)$.

We also apply our algorithm to a finer mesh, with 40×16 design variables (Fig. 5a). SOLO-G converges at $n_{train} = 1,912$, shown in Fig. 5b. Our design (Fig. 5d, $\widetilde{P} = 0.8062$) is found to be better than the solution from the gradient-based algorithm (Fig. 5c, $\widetilde{P} = 0.8065$). Intermediate solutions from SOLO-G are shown in Supplementary Fig. 5. Compared with directly using BBA which needs 2×10^8 evaluations, SOLO-G reduces the number of FEM calculations to $1/10^5$. It is interesting to note that the optimum in Fig. 5d has two gaps at the 7th and 12th columns. It is a little counterintuitive, since the gradient-based method gives a smooth ramp (Fig. 5c). We try filling the gaps and find that their existence indeed reduces pressure (see Supplementary Fig. 6), which demonstrates how powerful our gradient-free method is.

Conclusions and discussions

Topology optimization is an important problem with broad applications in many scientific and engineering disciplines. Solving non-linear high-dimensional optimization problems require gradient-free methods, but the high computational cost is a major challenge. We proposed an approach of self-directed online learning optimization (SOLO) to dramatically accelerate the optimization process and make solving complex optimization problems possible.

We demonstrated the effectiveness of the approach in solving compliance minimization problems and fluid-structure optimization problems. For the compliance problems with 25 and 121 continuous design variables, our approach converged and produced optimized solutions same as the known optimum with only 501 and 10,243 FEM calculations, respectively, which are about 1/400 of directly using GSA and FEM instead of DNN based on our estimation. For the fluid problems with 160 and 640 binary variables, our method (SOLO-G) converged after 286 and 1,912 FEM calculations, respectively, with solutions better than the benchmark. It used less than $1/10^5$ of FEM calculations compared with directly applying BBA to FEM, and converged much faster than another work based on reinforcement learning. Although overhead computation was introduced similar to other SMBO methods, it was almost negligible (see the time profile in Supplementary Table 1) and thus led to $2 \sim 5$ orders of magnitude of computation reduction compared with directly using heuristic algorithms. We expect the improvement of our approach is even larger considering the fact that heuristic methods may need multiple initializations and our approach can reveal abnormal solutions by monitoring the outputs.

Our algorithm is neat and efficient. As an amazing property observed from the tests, the number of function evaluations required by the approach does not grow exponentially as other heuristic

methods. Thus, it has great potential for large-scale applications. We bring a new perspective for high-dimensional optimization by embedding deep learning in optimization methods. More techniques, such as parallel FEM computation, uncertainty modeling and disturbance based on sensitivity analysis, can be incorporated to enhance the performance.

Methods

Enforcement of volume constraint. In the two compliance problems, all matrices representing the density distribution ρ have the same weighted average $\sum_{i=1}^{N} w_i \rho_i = V_0$ due to the volume constraint where w_i denotes the weight of linear Gaussian quadrature. A matrix from the initial batch is generated by three steps:

1. Generate a random matrix with elements uniformly distributed from 0 to 1.
2. Rescale the array to enforce the predefined weighted average.
3. Set the elements greater than one, if any, to 1 and then adjust those elements less than one to maintain the average.

Matrices for the second batch and afterwards add random disturbance to optimized solutions $\widehat{\rho}$ and then go through *Step 2* and *3* above to make sure the volume satisfies the constraint.

Finite Element Method (FEM). The energy and pressure of material distribution design are calculated by FEM as the ground truth to train the DNN. The meshes of FEM are the same as the design variables. Numerical results are obtained by COMSOL Multiphysics 5.4. Solutions from gradient-based methods are also obtained by COMSOL with optimality tolerance as 0.001. For the fluid problems, the gradient-based method produces a continuous array, and we use multiple thresholds to convert it to binary arrays and recompute their objective (pressure) to select the best binary array.

Deep Neural Network (DNN). The structure of the DNN used in this paper is presented in Fig. 6. There are three hidden layers attached with two dropout layers, one between Layer 2 and Layer 3 and the other between Layer 3 and the Output Layer. The input 2D matrix is flattened to a 1D vector as the input to DNN. All inputs are normalized before training and we introduce batch normalization (BN)[32] within the network as regularization. The output of DNN is reciprocal of energy or pressure to give better resolution at lower energy or pressure. (For the rest of this paper including Fig. 6, we regard the DNN to approximate energy or pressure for simplicity.) To optimize the DNN training process, we apply the ADAM[33] as the optimizer implemented on the platform of PyTorch 1.2.0[34]. The learning rate is 0.01. The loss function is set as Mean Square Error (MSE)[35]. All models are trained for 1,000 epochs with a batch size of 1,024 (if the number of training data is less than 1,024, all the data will be used as one batch).

Mutation and crossover. After calculating the optimized array $\widehat{\rho}$, more training data are generated by adding disturbance to it. There are two kinds of disturbance, as shown in Fig. 7.

Mutation means mutating several adjacent cells in the optimized array, i.e., generating random numbers from 0 to 1 to replace the original elements. In the 2D example shown in Fig. 7a, the

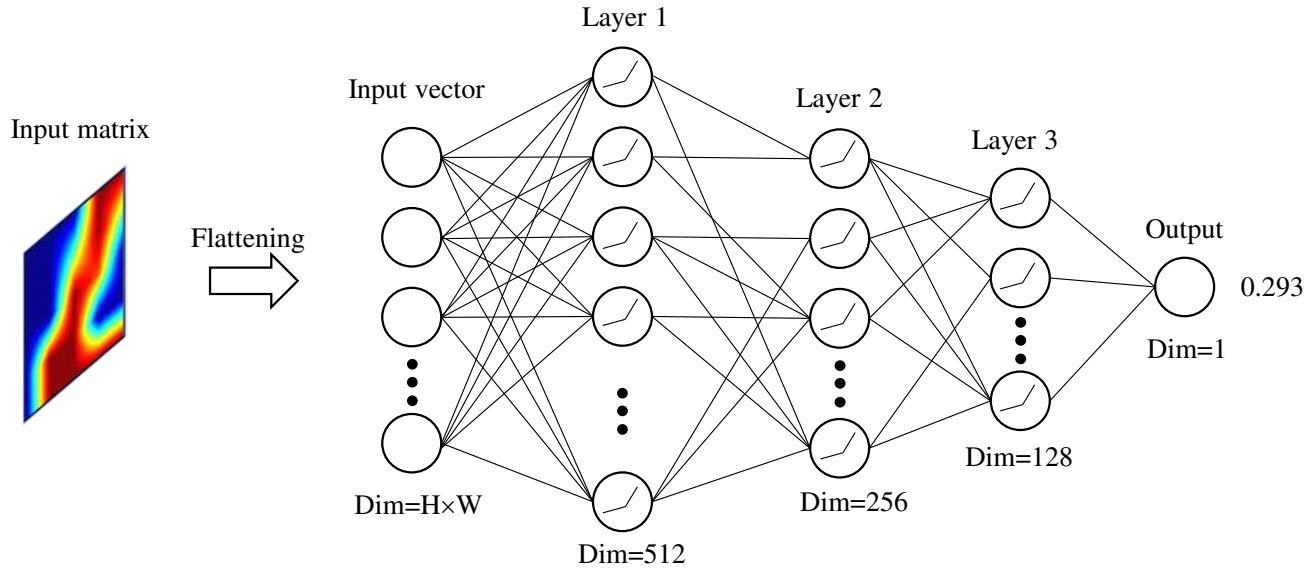

Fig. 6: Structure of the DNN

numbers in a 2-by-2 box are set as random. Mutation is likely to change the weighted average of the array, so the enforcement of volume constraint is applied after mutation.

Crossover, different from the genetic algorithm, denotes the crossover of cells in the array $\widehat{\rho}$, is achieved by the following steps:

1. Assign a linear index to each element in the array.
2. Randomly pick several indices.
3. Generate a random sequence of the indices.
4. Replace the original numbers according to the sequence above. As shown in Fig. 7b, indices are assigned sequentially from left to right and from top to bottom. The indices we pick in *Step 2* are 3, 4 and 8; the sequence generated in *Step 3* is 4, 8 and 3. Then the enforcement of volume constraint is applied.

In the two compliance minimization problems, the ways to generate a new input matrix based on $\widehat{\rho}$ and their possibilities are:

- mutating one element in $\widehat{\rho}$ (10%);
- mutating a 2×2 matrix in $\widehat{\rho}$ (10%);
- mutating a 3×3 matrix in $\widehat{\rho}$ (20%);
- mutating a 4×4 matrix in $\widehat{\rho}$ (20%);
- choosing an integer n from one to the number of total elements, selecting n cells in $\widehat{\rho}$ and exchanging them (20%);
- generating a completely random matrix like the initial batch (20%).

In the fluid-structure optimization problem with 20×8 mesh, the ways are the same as previous ones except a threshold is needed to convert the continuous array into a binary one. The threshold has 50% probability to be β^4 where β is uniformly sampled from [0,1], and has 50% probability

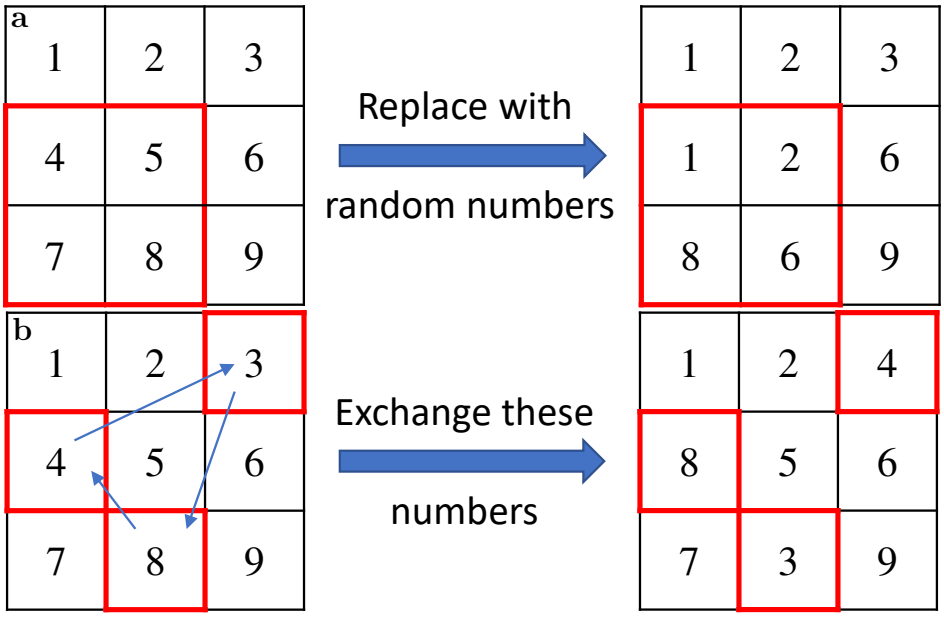

Fig. 7: Illustration of mutation and crossover. **a**, An example of mutation: some adjacent cells (in the red box) are replaced with random numbers. **b**, An example of crossover: several cells (in the red boxes) are exchanged. The volume constraint will be enforced at next step, not shown here.

to be the element-wise mean of $\widehat{\rho}$.

Generalized Simulated Annealing (GSA). Simulated Annealing (SA) is a stochastic method to determine the global minimum of a objective function by simulating the annealing process of a molten metal[36]. GSA is a type of SA with specific form of visiting function and acceptance probability[37]. Assuming objective

$$\widehat{\rho} = \underset{\rho \in [0,1]^N}{\arg\min}\ h(\rho), \tag{10}$$

we do the following:

1. Generate an initial state $\rho^{(0)} = (\rho_1^{(0)}, \rho_2^{(0)}, ..., \rho_N^{(0)})$ randomly and obtain its function value $E^{(0)} = f(\rho^{(0)})$. An initial temperature $T(0) = 5230$ is set. $imax$ is set to be 1000.
2. For artificial time step $t = 1$ to $imax$,
   (a) Generate a new state $\rho^{(i)} = \rho^{(i-1)} + \Delta\rho$, where $\Delta\rho$ follows the visiting function

$$g(\Delta\rho(t)) \propto \frac{[T(t)]^{-\frac{N}{3-q_v}}}{\left\{1 + (q_v - 1)\frac{[\Delta\rho(t)]^2}{[T(t)]^{\frac{2}{3-q_v}}}\right\}^{\frac{1}{q_v-1}+\frac{N-1}{2}}}. \tag{11}$$

   where q_v denotes a parameter set as 2.6 here and T denotes the artificial temperature calculated by

$$T(t) = T(0)\frac{2^{q_v-1} - 1}{(1+t)^{q_v-1} - 1}. \tag{12}$$

(b) Calculate the energy difference

$$\Delta E = E^{(i)} - E^{(i-1)} = h(\rho^{(i)}) - h(\rho^{(i-1)}). \tag{13}$$

(c) Calculate the probability to accept the new state

$$p = \min\left\{1, \left[1 - (1-q_a)\frac{t}{T(t)}\Delta E\right]^{\frac{1}{1-q_a}}\right\}, \tag{14}$$

where q_a is a constant set to be -5. Determine whether to accept the new state based on the probability, if not, $\rho^{(i)} = \rho^{(i-1)}$.

3. Conduct local search to refine the state.

The objective function used in the optimization process is written as

$$h(\rho) = f(\rho) + c(w \cdot \rho - V_0)^2, \tag{15}$$

where c is a constant to transform the constrained problem to an unconstrained problem by adding a penalty term. GSA is implemented via SciPy package with default parameter setting. For more details please refer to its documentation[38].

Binary Bat Algorithm (BBA). Bat Algorithm (BA) is a heuristic optimization algorithm, inspired by the echolocative behavior of bats. This algorithm carries out the search process using artificial bats mimicking the natural pulse loudness, emission frequency and velocity of real bats. Binary Bat Algorithm[39,40] is a binary version of BA. To solve

$$\widehat{\rho} = \underset{\rho \in \{0,1\}^N}{\arg\min}\ f(\rho), \tag{16}$$

we slightly adjust the original algorithm and implement it as follows:

1. Generate M vectors $\rho^{(0,1)}, \rho^{(0,2)}, ..., \rho^{(0,M)}$. We use $\rho^{(t,m)}$ to denote a vector, flattened from the array representing design variables. It is treated as the position of the m-th artificial bat, where $m = 1, 2, ..., M$. We use $\rho_i^{(t,m)} \in \{0, 1\}$ to denote the i-th dimension of vector $\rho^{(t,m)}$, where $i = 1, 2, ..., N$. Thus, $\rho^{(0,m)} = (\rho_1^{(0,m)}, \rho_2^{(0,m)}, ..\rho_N^{(0,m)})$.
2. Calculate their function values and find the minimum $\rho^* = \arg\min f(\rho^{(0,m)})$
3. Initialize their velocity $v^{(0,1)}, v^{(0,2)}, ..., v^{(0,m)}, ..., v^{(0,M)}$.
4. Determine parameters q_{min}, q_{max}, $imax$, α, $r^{(0)}$, $A^{(0)}$.
5. For artificial time step $t = 1$ to $imax$,
   (a) Update parameters $A^{(t)} = \alpha A^{(t-1)}$, $r^{(t)} = r^{(0)}(1 - e^{-\gamma t})$.
   (b) For $m = 1, 2, ..., M$,
      i. Calculate sound frequency

$$q^{(t,m)} = q_{min} + (q_{max} - q_{min})\beta, \tag{17}$$

where β is a random number that has a uniform distribution in [0,1].

      ii. Update velocity based on frequency

$$v^{(t,m)} = v^{(t-1,m)} + (\rho^{(t-1,m)} - \rho^*)q^{(t,m)}. \tag{18}$$

iii. Calculate the possibility to change position based on velocity

$$V^{(t,m)} = \left| \frac{2}{\pi} \arctan\left(\frac{\pi}{2} v^{(t,m)} \right) \right| + \frac{1}{N}. \tag{19}$$

iv. Generate $\beta_i'(i = 1, 2, ..., N)$, a series of random numbers uniformly in [0,1]. For those i satisfying $\beta_i' < V^{(t,m)}$, change the position by flipping the 0/1 values

$$\rho_i^{(t,m)} = 1 - \rho_i^{(t-1,m)}. \tag{20}$$

For others, keep them as they are.

v. Generate $\beta_i''(i = 1, 2, ..., N)$, a series of random numbers uniformly in [0,1]. For those i satisfying $\beta_i'' > r^{(t)}$, set $\rho_i^{(t,m)} = \rho_i^*$.

vi. Reverse to the previous step $\rho^{(t,m)} = \rho^{(t-1,m)}$, if $f(\rho^{(t,m)}) > f(\rho^{(t-1,m)})$ or $\beta''' > A^{(t)}$ (where β''' is random number uniformly in [0,1]).

(c) Update $\rho^* = \arg\min f(\rho^{(t,m)})$.

6. Output $\widehat{\rho} = \rho^*$.

Since we do not have constraint in the fluid problems, we can optimize f without adding penalty terms.

Code availability

All code (MATLAB and Python) used in this paper is available at `https://github.com/deng-cy/deep_learning_topology_opt`.

References

[1] Deaton, J. D. & Grandhi, R. V. A survey of structural and multidisciplinary continuum topology optimization: post 2000. *Structural and Multidisciplinary Optimization* **49**, 1–38 (2014).

[2] Bendsøe, M. P. & Kikuchi, N. Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering* **71**, 197–224 (1988).

[3] Rozvany, G. I. A critical review of established methods of structural topology optimization. *Structural and multidisciplinary optimization* **37**, 217–237 (2009).

[4] Sigmund, O. & Maute, K. Topology optimization approaches. *Structural and Multidisciplinary Optimization* **48**, 1031–1055 (2013).

[5] Hajela, P. & Lee, E. Genetic algorithms in truss topological optimization. *International Journal of Solids and Structures* **32**, 3341–3357 (1995).

[6] Shim, P. Y. & Manoochehri, S. Generating optimal configurations in structural design using simulated annealing. *International journal for numerical methods in engineering* **40**, 1053–1069 (1997).

[7] Kaveh, A., Hassani, B., Shojaee, S. & Tavakkoli, S. Structural topology optimization using ant colony methodology. *Engineering Structures* **30**, 2559–2565 (2008).

[8] Luh, G.-C. & Lin, C.-Y. Structural topology optimization using ant colony optimization algorithm. *Applied Soft Computing* **9**, 1343–1353 (2009).

[9] Luh, G.-C., Lin, C.-Y. & Lin, Y.-S. A binary particle swarm optimization for continuum structural topology optimization. *Applied Soft Computing* **11**, 2833–2844 (2011).

[10] Lee, K. S. & Geem, Z. W. A new structural optimization method based on the harmony search algorithm. *Computers & Structures* **82**, 781–798 (2004).

[11] Georgiou, G., Vio, G. A. & Cooper, J. E. Aeroelastic tailoring and scaling using Bacterial Foraging Optimisation. *Structural and Multidisciplinary Optimization* **50**, 81–99 (2014).

[12] Sigmund, O. On the usefulness of non-gradient approaches in topology optimization. *Structural and Multidisciplinary Optimization* **43**, 589–596 (2011).

[13] Bartz-Beielstein, T. A survey of model-based methods for global optimization. *Bioinspired Optimization Methods and Their Applications* 1–18 (2016).

[14] Hutter, F., Hoos, H. H. & Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, 507–523 (Springer, 2011).

[15] Frazier, P. I. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811* (2018).

[16] Hansen, N. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772* (2016).

[17] Jin, J., Yang, C. & Zhang, Y. An improved cma-es for solving large scale optimization problem. In *International Conference on Swarm Intelligence*, 386–396 (Springer, 2020).

[18] Wang, Z., Hutter, F., Zoghi, M., Matheson, D. & de Feitas, N. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research* **55**, 361–387 (2016).

[19] Lei, X., Liu, C., Du, Z., Zhang, W. & Guo, X. Machine Learning Driven Real Time Topology Optimization under Moving Morphable Component (MMC)-Based Framework. *Journal of Applied Mechanics* **86**, 011004 (2018).

[20] Banga, S., Gehani, H., Bhilare, S., Patel, S. & Kara, L. 3d topology optimization using convolutional neural networks. *arXiv preprint arXiv:1808.07440* (2018).

[21] Oh, S., Jung, Y., Kim, S., Lee, I. & Kang, N. Deep Generative Design: Integration of Topology Optimization and Generative Models. *Journal of Mechanical Design* 1–22 (2019).

[22] Sosnovik, I. & Oseledets, I. Neural networks for topology optimization. *Russian Journal of Numerical Analysis and Mathematical Modelling* **34**, 215–223 (2019).

[23] Rawat, S. & Shen, M.-H. H. A novel topology optimization approach using conditional deep learning. *arXiv preprint arXiv:1901.04859* (2019).

[24] Jang, S. & Kang, N. Generative design by reinforcement learning: Maximizing diversity of topology optimized designs. *arXiv preprint arXiv:2008.07119* (2020).

[25] Shen, M.-H. H. & Chen, L. A new cgan technique for constrained topology design optimization. *arXiv preprint arXiv:1901.07675* (2019).

[26] Yu, Y., Hur, T., Jung, J. & Jang, I. G. Deep learning for determining a near-optimal topological design without any iteration. *Structural and Multidisciplinary Optimization* **59**, 787–799 (2019). 1801.05463.

[27] Gaymann, A. & Montomoli, F. Deep neural network and monte carlo tree search applied to fluid-structure topology optimization. *Scientific reports* **9**, 1–16 (2019).

[28] Whitley, D. A genetic algorithm tutorial. *Statistics and Computing* **4**, 65–85 (1994).

[29] Bendsoe, M. P. & Sigmund, O. *Topology Optimization: Theory, Methods and Applications* (Springer, 2004).

[30] Deng, C., Qi, X. & Liu, Y. Numerical study on equilibrium stability of objects in fluid flow—a case study on constructal law. *Case Studies in Thermal Engineering* **15**, 100539 (2019).

[31] Olesen, L. H., Okkels, F. & Bruus, H. A high-level programming-language implementation of topology optimization applied to steady-state navier–stokes flow. *International Journal for Numerical Methods in Engineering* **65**, 975–1001 (2006).

[32] Ioffe, S. & Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[33] Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[34] Paszke, A. *et al.* Automatic differentiation in pytorch. In *NIPS-W* (2017).

[35] Lehmann, E. & Casella, G. *Theory of Point Estimation* (Springer Verlag, 1998).

[36] Xiang, Y., Gubian, S. & Martin, F. Generalized simulated annealing. In Peyvandi, H. (ed.) *Computational Optimization in Engineering*, chap. 2 (IntechOpen, Rijeka, 2017). URL https://doi.org/10.5772/66071.

[37] Xiang, Y., Gubian, S., Suomela, B. & Hoeng, J. Generalized Simulated Annealing for Global Optimization: The GenSA Package. *The R Journal* **5**, 13 (2013).

[38] The SciPy community. scipy.optimize.dual_annealing – Scipy v1.3.0 Reference Guide (2019). URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.dual_annealing.html.

[39] Mirjalili, S., Mirjalili, S. M. & Yang, X.-S. Binary bat algorithm. *Neural Computing and Applications* **25**, 663–681 (2014).

[40] Ramasamy, R. & Rani, S. Modified binary bat algorithm for feature selection in unsupervised learning. *Int. Arab J. Inf. Technol.* **15**, 1060–1067 (2018).

Acknowledgement

The authors gratefully acknowledge the support by the National Science Foundation under Grant No. CNS-1446117.

Author contributions

C.D. designed the algorithm and drafted the manuscript. Y.W. derived the convergence theory. C.D. and C.Q. wrote the code. Y.W and C.Q. edited the manuscript. W.L. supervised this study and revised the manuscript.

Figures

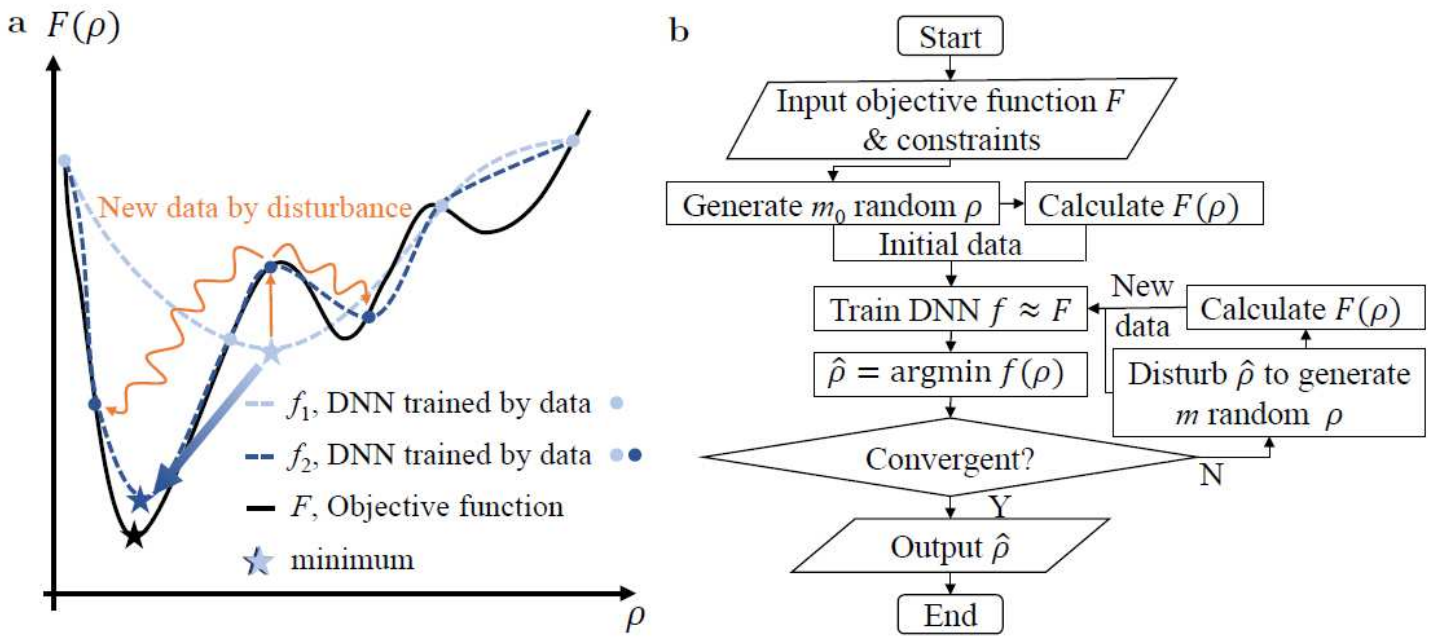

Figure 1

Schematics of the proposed self-directed online learning optimization. a, Schematic illustration of self-directed online training. The initial batch of training data (light blue dots) is randomly located. The DNN f1 (dashed light-blue line) trained on first batch of data only gives a rough representation of the true objective function F (solid black line). The second batch training data (dark blue dots) are generated by adding disturbance (orange curve) to the minimum of f1. After trained with two batches, the DNN f2 (dashed dark-blue line) is more refined around the minimum (the region of interest), while remains almost the same at other locations such as the right convex part. f2 is very close to finding the exact global minimum point. b, Flow diagram of the algorithm.

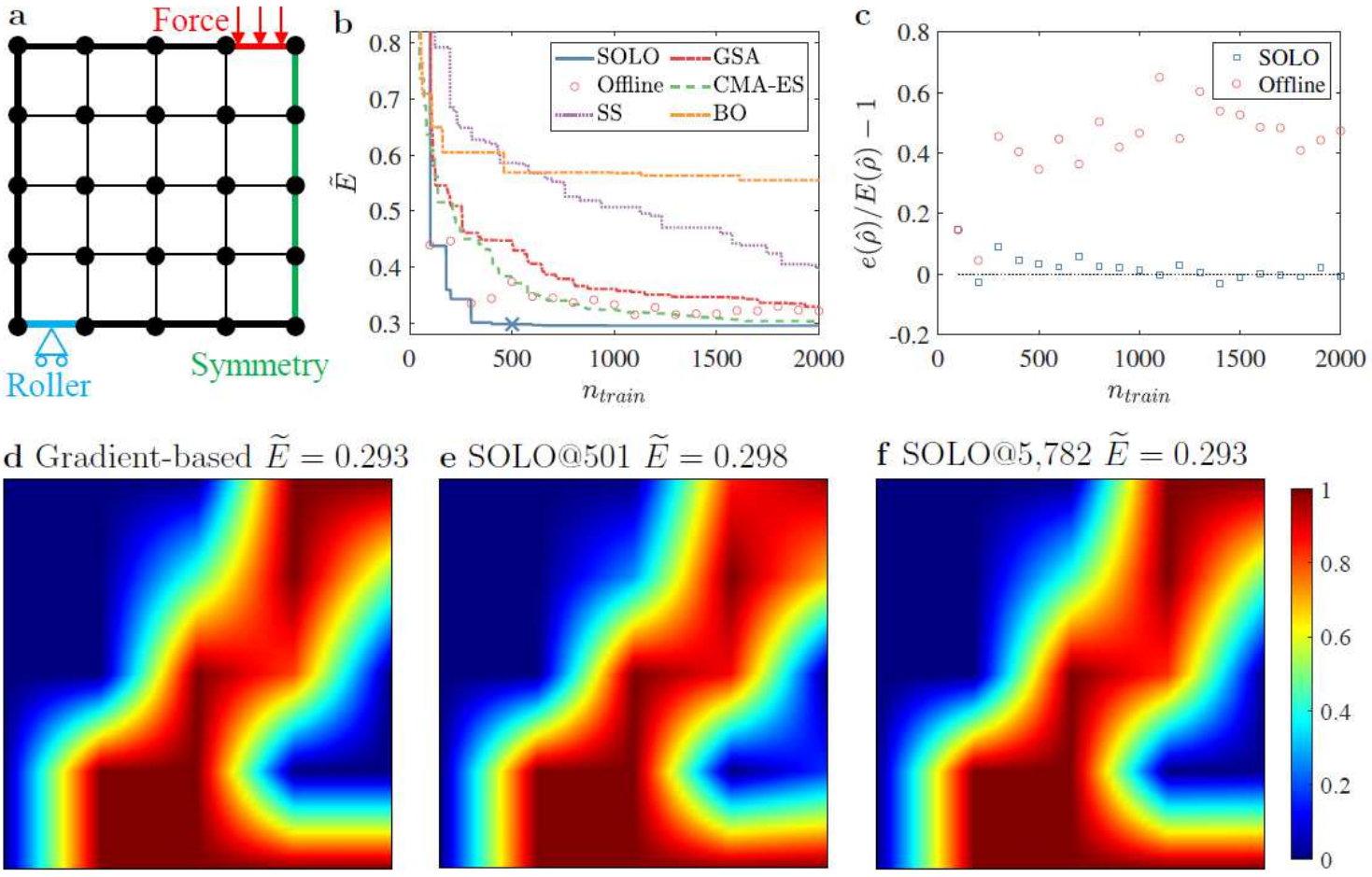

Figure 2

Setup and results of a compliance minimization problem with 5×5 design variables. a, Problem setup. b, Best dimensionless energy with a total of ntrain accumulated training samples. SOLO denotes our proposed method where the cross “X” denotes the con-vergence point (presented in e), “Offline” denotes training a DNN offline and then uses GSA to search for the optimum without updating the DNN, SS denotes Stochastic Search, which is the same as SOLO except that ρ in each loop is obtained by the minimum of existing samples, CMA-ES denotes Covariance Matrix Adaptation Evolution Strategy, BO denotes Bayesian Op-timization. SOLO converges the fastest among these methods. c, Energy prediction error of ρ relative to FEM calculation of the same material distribution. d, Optimized design produced by the gradient-based method. E = 0.293. e, Optimized design produced by SOLO. ntrain = 501 and E = 0.298. f, Optimized design produced by SOLO. ntrain = 5, 782 and E = 0.293. In d-f, dark red denotes ρ = 1 and dark blue denotes ρ = 0, as indicated by the right color scale bar.

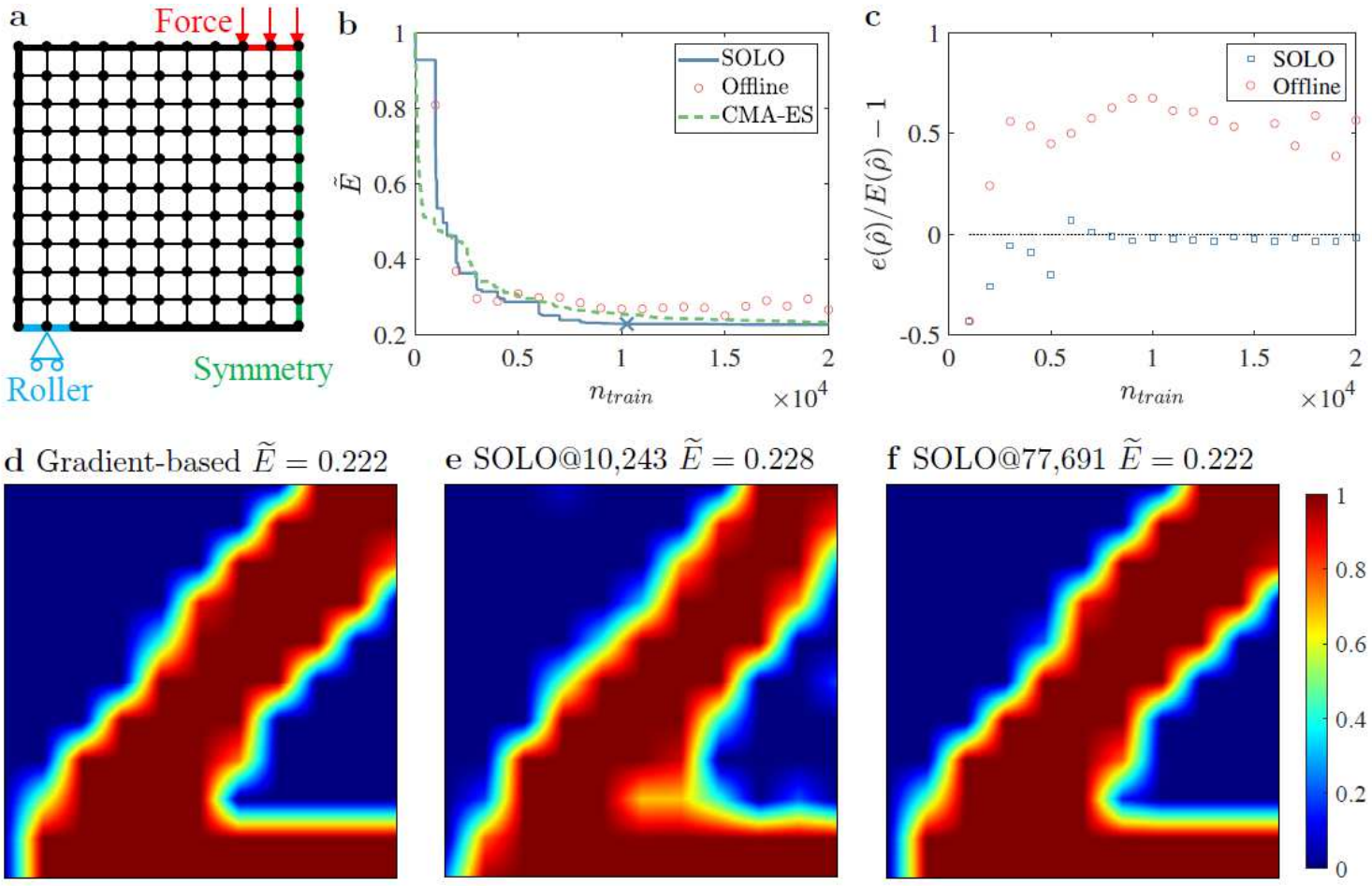

Figure 3

Setup and results of a compliance minimization problem with 11×11 design variables. a, Problem setup. b, Best dimensionless energy with a total of ntrain accumulated training samples. SOLO denotes our proposed method where the cross “X” denotes the con-vergence point (presented in e), “Offline” denotes training a DNN offline and then uses GSA to search for the optimum without updating the DNN, CMA-ES denotes Covariance Matrix Adaptation Evolution Strategy. SOLO converges the fastest among these methods. c, Energy prediction error of ρ relative to FEM calculation of the same material distribution. d, Optimized design produced by the gradient-based method. E = 0.222. e, Optimized design produced by SOLO. ntrain = 10, 243 and E = 0.228. f, Optimized design produced by SOLO. ntrain = 77, 691 and E = 0.222. In d-f, dark red denotes ρ = 1 and dark blue denotes ρ = 0, as indicated by the right bar.

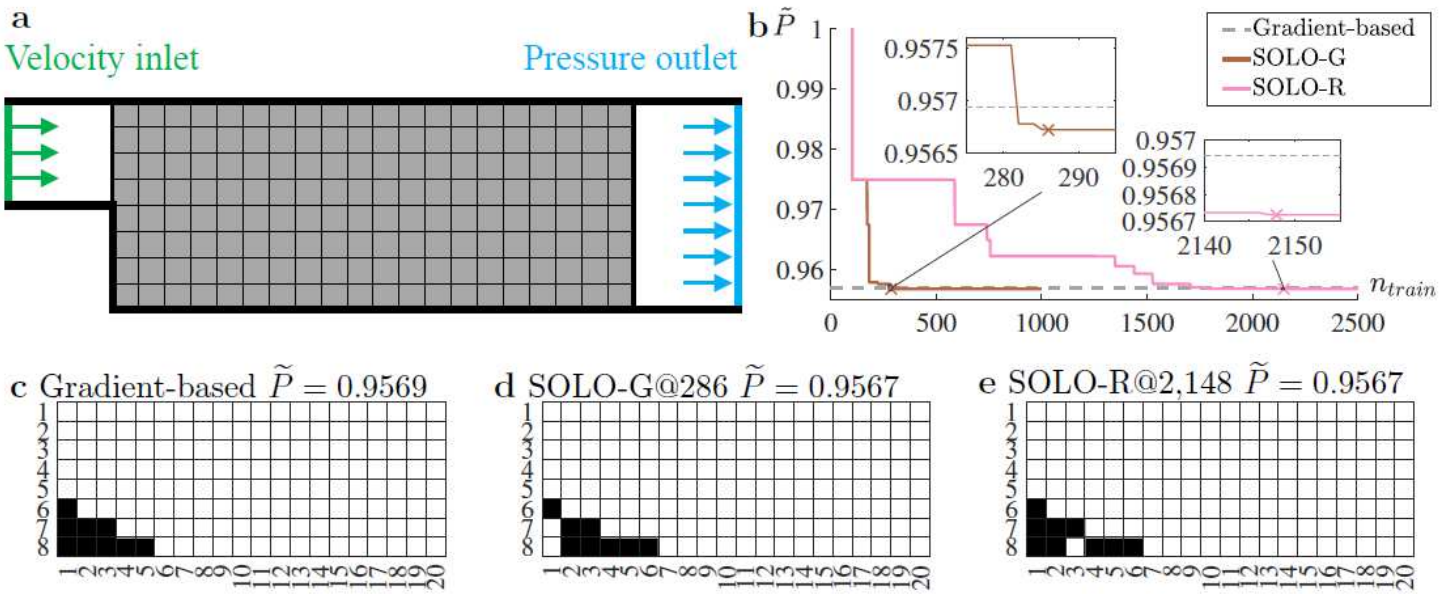

Figure 4

Setup and results of a fluid-structure optimization problem with 20×8 design variables. a, Problem setup. The vertical green line denotes the inlet while the vertical blue line denotes the outlet. b, Dimensionless inlet pressure versus ntrain, the number of accumulated training samples. SOLO-G denotes a greedy version of our proposed method, SOLO-R denotes the regular version of our proposed method. The horizontal dashed line denotes the solution from the gradient-based method. The cross “X” denotes the convergence point (presented in d and e, respectively). c, Optimized design obtained by the gradient-based method. P = 0.9569. d, Optimized design obtained by SOLO-G. ntrain = 286 and P = 0.9567. e, Optimized design obtained by SOLO-R. ntrain = 2, 148 and P = 0.9567. In c-e, black denotes ρ = 1 (solid) and white denotes ρ = 0 (void). These solutions are equivalent since the flow is blocked by the black squares forming the ramp surface and the white squares within the ramp at the left bottom corner are irrelevant.

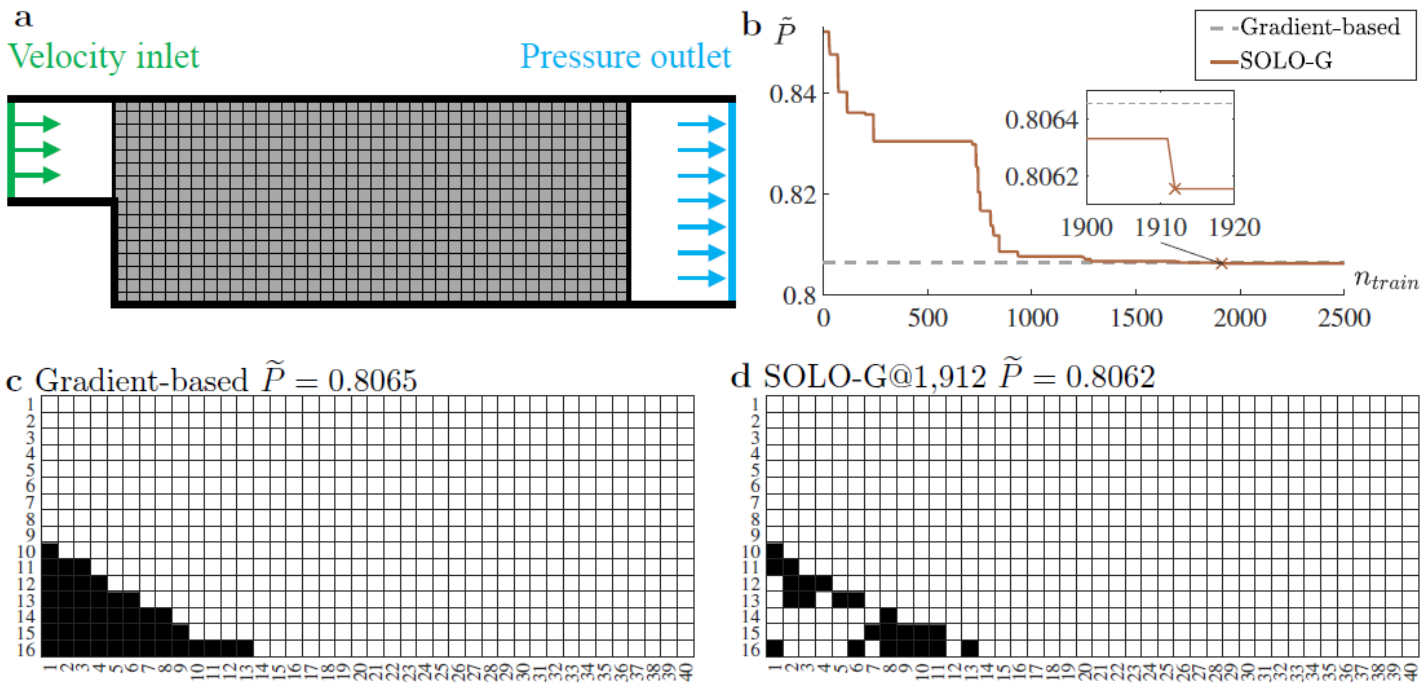

Figure 5

Setup and results of a fluid-structure optimization problem with 40×16 design variables. a, Problem setup. b, Dimensionless inlet pressure versus ntrain, the number of accumulated training samples. SOLO-G denotes a greedy version of our proposed method, where the cross “X” denotes the convergence point (presented in d). The horizontal dashed line denotes the solution from the gradient-based method. c, Optimized design obtained by the gradient-based method. P = 0.8065. d, Optimized design obtained by SOLO-G. ntrain = 1, 912 and P = 0.8062. In c,d, black denotes ρ = 1 (solid) and white denotes ρ = 0 (void). The SOLO-G result in d has two gaps at the 7th and 12th columns, while the gradient-based result in c gives a smooth ramp. We try filling the gaps and find that their existence indeed reduces pressure, which demonstrates the powerfulness of our gradient-free method.

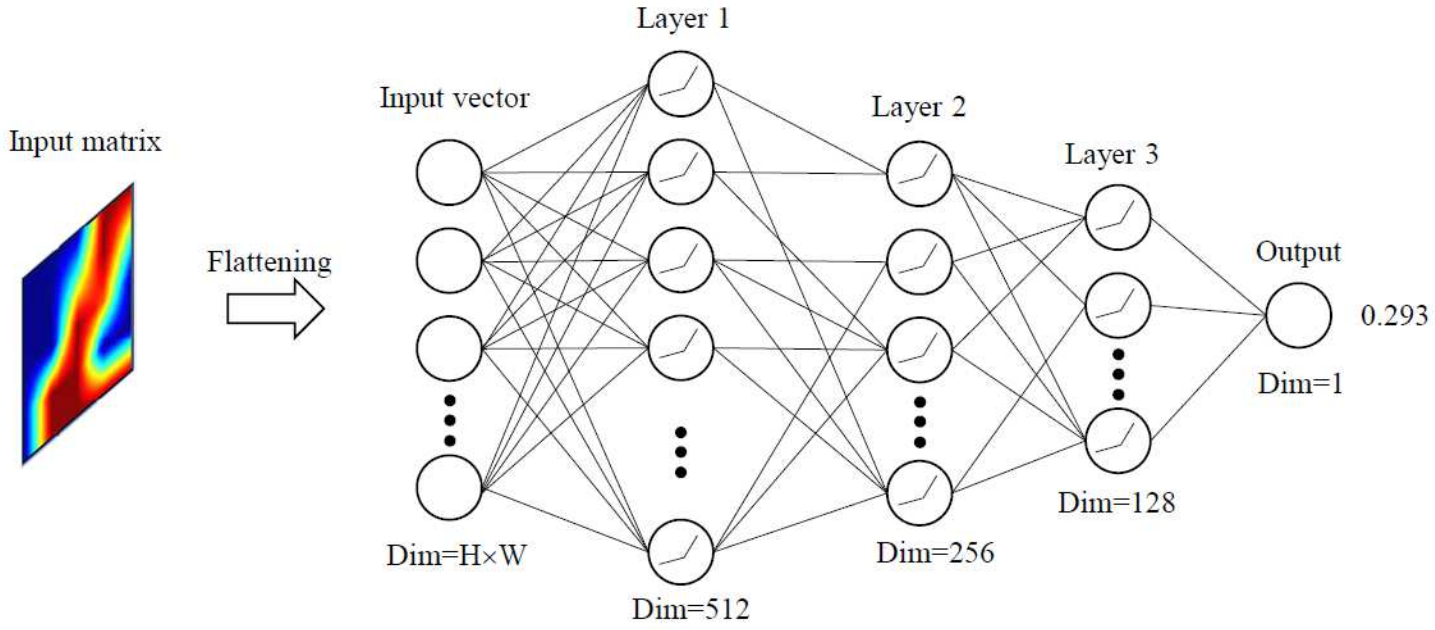

Figure 6

Structure of the DNN

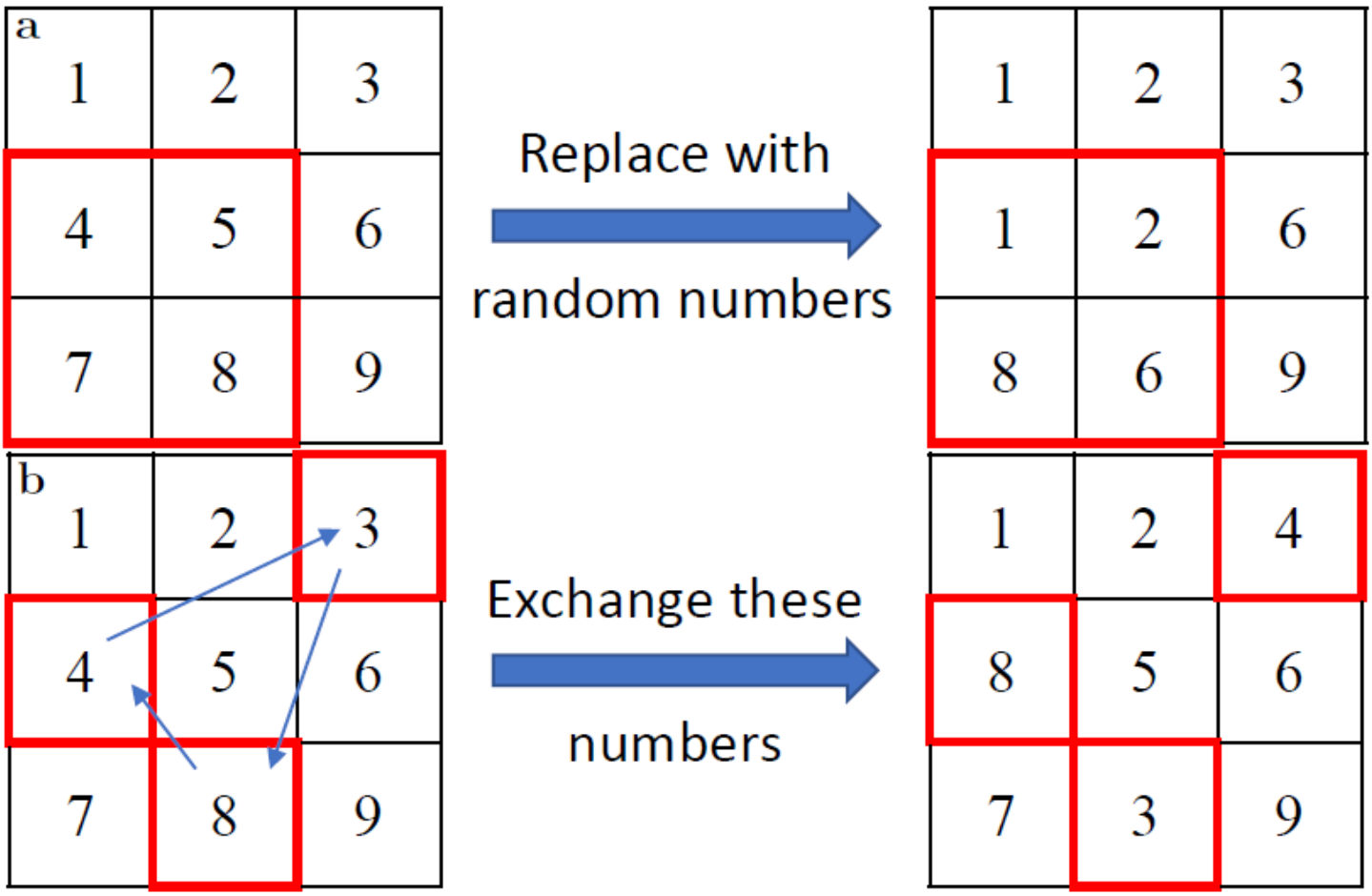

Figure 7

Illustration of mutation and crossover. a, An example of mutation: some adjacent cells (in the red box) are replaced with random numbers. b, An example of crossover: several cells (in the red boxes) are exchanged. The volume constraint will be enforced at next step, not shown here.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- SI.pdf