Neural optimization machine: a neural network approach for optimization and its application in additive manufacturing with physics-guided learning

Jie Chen¹ and Yongming Liu²

¹Department of Mechanical Engineering, Northwestern University, Evanston, IL 60208, USA
²School for Engineering of Matter, Transport, and Energy, Arizona State University, Tempe, AZ 85287, USA

Neural networks (NNs) are increasingly used in design to construct the objective functions and constraints, which leads to the needs of optimization of NN models with respect to design variables. A Neural Optimization Machine (NOM) is proposed for constrained single/multi-objective optimization by appropriately designing the NN architecture, activation function and loss function. The NN’s built-in backpropagation algorithm conducts the optimization and is seamlessly integrated with the additive manufacturing (AM) process-property model. The NOM is tested using several numerical optimization problems. It is shown that the increase in the dimension of design variables does not increase the computational cost significantly. Next, a brief review of the physics-guided machine learning model for fatigue performance prediction of AM components is given. Finally, the NOM is applied to design processing parameters in AM to optimize the mechanical fatigue properties through the physics-guided NN under uncertainties. One novel contribution of the proposed methodology is that the constrained process optimization is integrated with physics/knowledge and the data-driven AM process-property model. Thus, a
A physics-compatible process design can be achieved. Another significant benefit is that the training and optimization are achieved in a unified NN model, and no separate process optimization is needed.

This article is part of the theme issue ‘Physics-informed machine learning and its structural integrity applications (Part 1)’.

1. Introduction

Optimization is used in various scientific and engineering applications, such as optimal control, system planning, signal processing and mechanical design [1,2]. One possible and auspicious method for optimization is utilizing neural networks (NNs) due to their inherent massive parallelism [1], the ability to deal with time-varying parameters [3], the robustness of the computation [4] and a large community for rapid advancement in recent years [5,6].

Tank & Hopfield [7] proposed several neural optimization networks by applying circuit theory in optimization using NNs [4]. Dhingra & Rao [8] adapted Hopfield’s NN to solve nonlinear programming problems with/without constraints. The solutions from the NN approach agree well with those calculated by gradient-based search techniques. Wu et al. [9] presented a NN method for quadratic programming problems by applying the Lagrange multiplier theory. The solutions satisfy the necessary conditions for optimality. The connections in NNs are designed according to the optimization problems. Tagliarini et al. [10] presented a rule for NN design for optimization problems. Time evolution equations are constructed with equality and inequality constraints. Effati et al. [1] solved linear and quadratic programming problems using recurrent NNs. The NN approach establishes an energy function and a dynamic system. The solution can be found when the dynamic system approaches its static state. Xia et al. [3] developed a recurrent NN approach for optimization under nonlinear inequality constraints. That model was developed for convex optimization problems, and is suitable for only a specific category of non-convex optimization problems.

Another category of optimizing NNs is using the NN as a surrogate model. Neelakantan et al. [11] approximated the simulation model using the NN and applied nonlinear programming methods to find near-optimal policies. Nascimento et al. [12] replaced the equations in optimization problems using the NN. First, the grid search method is applied. Then, the solutions violating the constraints are excluded. This method allows one to identify multiple optima. Darvishvand et al. [13] trained NNs to construct the objective functions and then used Genetic Algorithm to find the optimal design variables. Jeon et al. [14] proposed a double-loop process for training and optimizing the NN objective function. The outer process is to optimize the NN weights. The inner process aims to optimize input variables while fixing the NN weights. Chandrasekhar et al. [15] used the NN as the density function for topology optimization. The inputs are location coordinates, and the outputs are density values. The density field is optimized using the NN’s backpropagation and a finite-element solver. Villarrubia et al. [16] used NNs to solve problems when the linear programming or Lagrange multiplier is not applicable. First, the NN is used for objective function approximation. Then the Lagrange method is used to solve optimization problems. However, the activation function is restricted to arctan due to its convenience in derivation for using Lagrange or Kuhn–Tucker conditions.

There are several limitations of current gradient-based methods for optimizing NN objective functions. The first limitation is that the architecture of the NN is restricted. That is, the activation function, types of layers (e.g. convolutional layer) and the structure of the NN need to be particular forms to enable the derivation of derivatives of NN outputs with respect to NN inputs. To overcome that issue, in this work, we use the NN built-in automatic differentiation so that no manual derivation of gradients is required. The second limitation is that the constraints need to be simple to use some nonlinear programming methods or Kuhn–Tucker conditions. In this
work, we propose to use the penalty function method. The constraints are incorporated in the loss function. For complex constraints or physics knowledge, they can be constructed using NNs. The third limitation is that the algorithm converges to one local optimum. We propose to utilize the randomness of NN training (i.e. random parameter initiation) combined with multiple starting points from grid search to find multiple local minima.

Traditionally optimization techniques are used to train the machine learning models. For example, particle swarm optimization was used to train NNs to obtain optimal weights and biases [17]. After training, the machine learning models are used as prediction models [18]. The similarities between the proposed method and the above-mentioned studies are that they both employed the flexibility of NN to describe the complex nonlinear relationships in scientific computing and modelling. The major difference is that the proposed Neural Optimization Machine (NOM) integrates the training portion and optimization portion in a single framework, which can directly use the intrinsic capabilities in NN (e.g. backpropagation for gradients calculation) and many other available algorithm developments in the package. In this paper, we proposed a new, alternative and systematic methodology to use NNs as a tool for optimizing the prediction models (i.e. NN models) to obtain the optimal design variables rather than NN parameters.

In this work, NN is used as a surrogate model of the objective function. The inputs of the NN are the design variables, and the outputs are the quantities to be optimized. This work aims to optimize the outputs of the NN objective function with respect to its inputs. This paper proposes a NN approach for optimizing NN surrogate models with and without constraints. NNs see the behaviour of the systems or signals and learn their behaviour to predict the future or recognize different features. In this work, we transform an optimization problem into an estimation problem. In other words, we solve an estimation problem that can optimize the system behaviour. The method is called NOM. NN models are used as the objective functions for the NOM. The optimization process is carried out by utilizing the NN’s built-in backpropagation algorithm. The architecture of the NN objective function is extended. The structure, activation function, and loss function of NOM are appropriately designed. The proposed NOM has the following benefits. First, the NOM is very flexible, and the NN objective function can have arbitrary architectures and activation functions. Second, the NOM is not limited to specific optimization problems, e.g. linear and quadratic programming. Third, multiple local minima can be found, which provides the potential for finding the global minimum. Fourth, compared with current heuristics optimization techniques, e.g. Particle Swarm Optimization and Genetic Algorithm, the increase of dimension of design variables does not increase the computational cost significantly. Fifth, the NOM can be easily extended and adapted for multi-objective optimization.

The remainder of the paper is organized as follows. First, a brief introduction to NNs is provided. Following this, the methodology of the NOM is presented. The construction of the NOM for unconstrained and constrained optimization is described. The NOM is then extended for multi-objective optimization. Next, the NOM is tested using numerical optimization problems and applied to the design in additive manufacturing. Next, a discussion is given regarding the NOM finding multiple local minima. Finally, several conclusions are drawn.

2. Neural optimization machine

(a) A brief introduction to NNs

Figure 1 shows an illustrative example of the main concepts of NNs [19–22]. There are three layers in this example. The first and the last layer are the input and the output layer, respectively. The layer in between is the hidden layer. The nodes in each layer are called neurons. This example has two neurons and one neuron at the first and last layer, respectively. Thus, it is a two-dimensional problem with one output. The NN in this example is called a feedforward NN since information flows layer by layer from the input layer to the output layer.
The mathematical calculation in each neuron can be expressed as follows [23]:

$$z^{(l)}_k = b^{(l-1)}_k + \sum_{j=1}^{p_{l-1}} w^{(l-1)}_{kj} a^{(l-1)}_j, \quad l = 2, \ldots, L$$  \hspace{1cm} (2.1)

and

$$a^{(l)}_k = c^{(l)}_k(z^{(l)}_k), \quad k = 1, 2, \ldots, p_l,$$  \hspace{1cm} (2.2)

where $l$ is the layer index. $l$ is 1 and $L$ for the input and output layers, respectively. The number of neurons at the $l$th is $p_l$. At the first layer (input layer),

$$a^{(1)}_k = x_k, \quad k = 1, \ldots, p_1.$$  \hspace{1cm} (2.3)

The $k$th neuron at layer $l$ has an activation function $G^{(l)}_k(\cdot)$. As shown in figure 1, $a$ is the output of a neuron, and $z$ is the linear combination of the outputs of all neurons at the previous layer. When the layer is input layer, $a$ becomes $x$, which is the input variable. The coefficients and intercepts in equation (2.1) in NNs are named weights and biases, respectively. During the NN training, weights and biases are updated while minimizing the loss function [24]. Training a NN is a process to minimize the loss function with respect to the NN parameters. The selection of the loss function depends on the goal of the NN. For NN as a surrogate model of the objective function to be optimized, the goal is to fit a regression model. For NOM, the goal is to optimize the outputs of the NN objective function. A common optimization technique in NNs is stochastic gradient descent. Mini-batches of the total training data train the NN to alleviate the computational burden. The training data are used in a mini-batch manner in each epoch until all the data are used. A number of epochs are needed to train a NN.

(b) Neural optimization machine

Let $NN(X)$ denote a trained NN model that is the objective function in optimization. The goal is to find the set of inputs $X$, which minimize the outputs of $NN(X)$ under inequality and equality constraints.
Figure 2. A simple neural network.

The formulation of the optimization under constraints is as follows,

\[
\begin{align*}
\min & \quad f(X) = \text{NN}(X) \\
\text{s.t.} & \quad g_p(X) \leq 0, \quad p = 1, \ldots, P \\
& \quad h_q(X) = 0, \quad q = 1, \ldots, Q,
\end{align*}
\]  

where \(f(X)\) is the objective function in the optimization problem. In this work, \(f(X)\) is the NN surrogate model. \(X\) is the vector of the design variables in \(f(X)\) and the input variables in \(\text{NN}(X)\), and \(g\) represent the inequality and \(h\) represent equality constraints. Figure 2 shows a simple neural network \(\text{NN}(X)\), which is the same as the architecture in figure 1.

The key idea of the developed NOM is to solve the optimization problem in equation (2.4) using the NN’s built-in backpropagation algorithm by properly designing the NN architecture. On the one hand, in the backpropagation algorithm, the basic method is stochastic gradient descent. It computes the gradients of the loss function with respect to the NN parameters (i.e. weights and biases). The weights and biases are then updated by the gradient information [25]. At the end of the training, a set of local optimal weights and biases is obtained. On the other hand, the gradient descent algorithm can also solve the optimization problem in equation (2.4). It requires computing the gradients of the outputs of the NN objective function with respect to its inputs. Then, the question is, \textit{can we transform optimizing the NN outputs with the NN inputs as design variables to optimizing NN loss function with NN parameters as design variables}? If this can be achieved, another question is \textit{how to consider the constraints in equation (2.4)}. The NOM is developed to solve those two issues.

\(\text{(i) NOM for unconstrained optimization}\)

To illustrate the basic components of the NOM, we will first consider the unconstrained optimization problem, i.e. the problem without the constraints in equation (2.4). The NOM architecture in figure 3 is designed to answer the first question, that is, to transform optimizing the NN outputs with the NN inputs as design variables to optimizing NN loss function with NN parameters as design variables. The subpart of the NOM architecture shown in the dashed line box in figure 3 is the trained NN objective function to be optimized in figure 2. It is called NN objective function to differentiate it from the NOM. A new layer (i.e. starting point layer) is added ahead of the input layer of the NN objective function. This example has two starting point neurons (as shown in grey) in this layer. Each neuron in the starting point layer is connected to one of the neurons in the input layer. There are no activation functions in the starting point layer. The values input to the NN objective function are determined by the starting points and
NN parameters from starting point layer to input layer. Following the formulation in equations (2.1)–(2.3), the input values are calculated by

$$z_k^{(1)} = b_k^{(s)} + w_k^{(s)} a_k^{(s)}, \quad k = 1, 2, \ldots, p_1$$

and

$$x_k = a_k^{(1)} = z_k^{(1)},$$

where the superscript $(s)$ indicates the starting point layer of the NOM, the number one in (1) indicates the first layer (i.e. the input layer) in the NN objective function and $p_1$ is the dimension of design variables. The optimal $x_k$, $k = 1, 2, \ldots, p_1$, are desired to minimize the NN objective function. They are obtained from equation (2.6) by training the NOM.

To achieve this goal, we customize the loss function of NOM and constrain the NN weights and biases. The output of the NOM is used as the NOM loss function. In this case, the NOM outputs are the same as the outputs of the NN objective function. In figure 3, different colours of the connections between neurons mean whether the NN parameters are fixed (orange) or trainable (blue). The NN parameters are fixed for the NN objective function (shown in the dashed line square) and are trainable from starting point layer to input layer. This is because when training the NOM, the original NN model (i.e. NN objective function) should be kept unchanged. At the same time, the NN parameters from starting point layer to input layer are updated to find the optimal solution to minimize the NOM.

Only one training data point is used to train the NOM. The data point is referred to as the starting point in stochastic gradient descent in optimizing the NOM. These data can also be interpreted as the training data for the input of the NOM (i.e. the starting point is the input to the starting point layer of the NOM). There are no training data for the output of NOM since the NOM is not trained in a supervised way.

A general drawback of NN training is the convergence to local minima. As a result, the NOM will produce a locally optimal solution. The following two operations are adopted to mitigate this issue and possibly find the global minimum. First, multiple starting points are generated inside the region of interest through grid search. The points with constraint violations are then excluded [12]. Next, each starting point is used as the training data for NOM to find the corresponding optimal solution to the NN objective function. Second, before training the NOM, random initial values are assigned to the NN parameters from starting point layer to input layer. This is a default feature of the optimizer in training the NN. This operation adds some randomness to the initial search directions. The effects of the above operations are further discussed in §5.

(ii) NOM for constrained optimization

The above description of the NOM shown in figure 3 is for unconstrained optimization. Based on the NOM for unconstrained optimization, the NOM is further developed and shown in figure 4...
Figure 4. NOM for constrained optimization.

for answering the second question, i.e. how to consider the constraints. Compared with the NOM in figure 3, a constraint layer is added. The neurons in this layer are called constraint neurons, as shown in green. Each constraint neuron represents a constraint in figure 3. Following equations (2.1) and (2.2), the calculation from input layer to constraint layer can be expressed as

\[ z_k^{(c)} = g_k(X), \quad k = 1, \ldots, P, \] (2.7)

for inequality constraints, and

\[ z_k^{(c)} = h_k(X), \quad k = P + 1, \ldots, P + Q, \] (2.8)

for equality constraints, and then

\[ a_k^{(c)} = G_k^{(c)}(z_k^{(c)}), \quad k = 1, 2, \ldots, P + Q, \] (2.9)

where the superscript \((c)\) indicates the constraint layer. We implement the penalty function method for constrained optimization. A constrained optimization is transformed to an unconstrained optimization by adding the violated constraints to the objective function as a penalty. The activation functions used for neurons in the constraint layer serve as a penalty function. The adopted activation function \(G^{(c)}\) for the constraint layer is modified from the Rectified Linear Unit (ReLU) activation function, which is expressed as

\[ \text{ReLU}(z) = \max(0, z). \] (2.10)

The inequality constraints in problem (4) is handled by the constraint neuron using the activation function

\[ G^{(c)}(z) = c \cdot \text{ReLU}(z), \] (2.11)

and the activation function for equality constraints are,

\[ G^{(c)}(z) = c \cdot [\text{ReLU}(-z) + \text{ReLU}(z)], \] (2.12)

where \(c\) controls the extent of penalty. The plots of ReLU (equation (2.10)) and two modified ReLU activation functions (equations (2.11) and (2.12)) with \(c = 10\) are plotted in figure 5. The physics meaning of using the above activation functions is that when the constraints are satisfied, the outputs of the constraint neurons are zero. When the constraints are violated, the outputs of the constraint neurons are large values.

Another difference between the NOM architectures in figures 3 and 4 is the NOM output. In figure 3, the outputs of the NOM and the NN objective function are the same. In figure 4, NOM outputs are calculated by summing outputs of the NN objective function and all values from
constraint neurons. The NOM loss function is still the NOM outputs. A large penalty value is added to the loss function when the constraints are violated. The NOM is trained to move the design variables to the feasible domain by adjusting the NN parameters from starting point layer to input layer.

A flowchart is shown in figure 6 to summarize the NOM methodology for constrained optimization. In summary, NOM aims to optimize a NN model. NOM architecture is built upon the NN objective function by adding starting point layer, constraint layer and NOM output layer. The weights and biases in the NN objective function and between the starting point layer and constraint layer are fixed during the NOM training. The loss function is the NOM output. Multiple starting points are used for finding the global optimal solution.

NNs are used as surrogate models of the objective function. The goal is to obtain the optimal input variables. Current research for optimizing a NN objective function can be categorized into direct search [12], heuristics methods [13] and gradient-based methods [11,14–16]. NOM belongs to the gradient-based methods. NOM used NN built-in automatic differentiation and optimizer to optimize the input variables. A significant difference between NOM and the above gradient-based methods is that NOM does not need manual derivation of gradients, which provides the flexibility of the NN architecture of the objective function. NN with any activation function or any type of NNs (e.g. convolutional NNs) can be optimized using NOM. Also, constraints are not restricted to any particular form in NOM. In addition, NOM is versatile and can be extended to solve problems with multiple objectives. To the best of the authors’ knowledge, this is the first attempt to formulate the problem in the way we proposed.

(c) NOM for multi-objective optimization

The proposed NOM is a flexible tool. It can be developed to solve other types of optimization problems involving NNs as objective functions. This section shows how the NOM can be developed for multi-objective optimization for NNs.

The multi-objective optimization problem is formulated as

\[
\begin{align*}
\min & \quad F(X) = \{f_1(X), f_2(X), \ldots, f_M(X)\} \\
\text{s.t.} & \quad g_p(X) \leq 0, \quad p = 1, \ldots, P \\
& \quad h_q(X) = 0, \quad q = 1, \ldots, Q,
\end{align*}
\]

where the \(M\) objective functions are \(M\) NN models. Usually, there is no \(X\) that can minimize all objective functions simultaneously [26]. Infinite solutions can be obtained for different combinations of good performance of objectives. Those solutions are called Pareto optimum.
Train neural networks (NNs) as objective functions

**Goal:** NN as a surrogate prediction model

**Build Neural Optimization Machine (NOM)**

**NOM architecture:** starting point layer, constraint layer, NN objective functions, NOM output layer

**Fix parameters (weights and biases) for NN objective functions, constraint layer**

**Loss function:** NOM output

**Train NOM with multiple starting points**

**Training data:** starting points

**Trainable parameters:** weights and biases between starting point layer and input layer

**Goal:** find optimal design variables

---

**Figure 6.** Flowchart of NOM for constrained optimization.

---

solutions [27–29]. A Pareto optimal is a feasible solution $X$ that satisfies the following condition: we cannot find another feasible solution $Y$ that $f_i(Y) \leq f_i(X)$ for all objectives ($i = 1, 2, \ldots, M$) and $f_j(Y) < f_j(X)$ for at least one objective function. [26]. The goal of multi-objective optimization is to find the Pareto optimum solution set.

Among several methods developed for multi-objective optimization, the bounded objective function method is used, which is suitable for the proposed NOM. Consider the maximum and minimum acceptable values for objective function $f_i$ are $U_i$ and $L_i$ ($i = 1, 2, \ldots, M$), respectively. We minimize the most important (say, the $r$th) objective function [26]:

$$
\begin{align*}
\min & \quad f_r(X) = NN_r(X) \\
\text{s.t.} & \quad g_p(X) \leq 0, \quad p = 1, \ldots, P \\
& \quad h_q(X) = 0, \quad q = 1, \ldots, Q \\
& \quad L^{(i)} \leq f_i(X) = NN_i(X) \leq U^{(i)}, \quad i = 1, \ldots, M, i \neq r.
\end{align*}
$$

(2.14)
$L^{(i)}$ can be discarded if the goal is not to achieve a solution falling within a range of values for each objective. In that case, $U^{(i)}$ can be varied systematically to produce the Pareto optimal solution set [30]. That is adopted in this paper. A general guideline for selecting $U^{(i)}$ is [31]

$$f_i(X_i^*) \leq U^{(i)} \leq f_r(X_r^*),$$

where $X_i^*$ is the optimal solution of $f_i(X)$. To find $X_i^*$ is a single objective optimization problem where NOM can be applied. More discussion on selecting $U^{(i)}$ can be found in [32,33].

A numerical example with two NN objective functions is used to illustrate the proposed NOM in multi-objective optimization. The NOM for multi-objective optimization is shown in figure 7. The subparts of the NOM in the two dashed line boxes are the two NN objective functions, $NN_1$ and $NN_2$, respectively. The architecture of each NN objective function is shown in figure 2. Compared with the NOM in figure 4, another NN objective function is added in figure 7. Both NN objective functions share the same input layer in this example. According to the bounded objective function method, $NN_1$ is used as the objective function, and $NN_2$ is used as the constraint. The output neuron of $NN_2$ is replaced by the constraint neuron. The upper bound, $U^{(2)}$, changes according to equation (2.15). For each $U^{(2)}$, a Pareto optimal solution is obtained using the NOM. Pareto optimal solutions can be obtained by varying the $U^{(2)}$.

3. Numerical experiments and model verification

In this section, the proposed NOM is tested for constrained and multi-objective optimization numerical problems. The objective is to verify the proposed NOM with benchmark methods using numerical experiments before its application to the additive manufacturing (AM) process optimization problem.
Table 1. The hyperparameters for training the NN objective functions.

| Hyperparameter                      | Value |
|-------------------------------------|-------|
| number of hidden layers             | 1     |
| number of hidden neurons            | 20    |
| the activation function of hidden layers | hyperbolic tangent (tanh) |
| epochs                              | 500   |
| minibatch size                      | 10    |
| learning rate                       | 0.01  |

Table 2. The hyperparameters for the NOM.

| Hyperparameter                      | Value |
|-------------------------------------|-------|
| number of starting points           | 5     |
| penalty parameters                  | 10    |
| epochs                              | 2000  |
| minibatch size                      | 1     |
| learning rate                       | 0.01  |

(a) Constrained optimization problems

In this subsection, three optimization problems are analysed. First, the results from the NOM are compared with those from Nelder Mead, Generic Algorithm, Differential Evolution and Particle Swarm Optimization.

The data used to train the NN objective function are generated inside the domain defined by the constraints on individual variables; 10,000 data are generated evenly for each problem. The hyperparameters for training the NN objective functions and the NOM are identical for all three problems and are listed in tables 1 and 2, respectively.

The first problem (Problem 1) has a cubic objective function with quadratic constraints:

\[
\begin{align*}
\min \quad & f(x_1, x_2) = \frac{1}{10000}[(x_1 - 10)^3 + (x_2 - 10)^3] \\
\text{s.t.} \quad & -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0 \\
& (x_1 - 6)^2 - (x_2 - 5)^2 - 100 \leq 0 \\
& 13 \leq x_1 \leq 20 \\
& 0 \leq x_2 \leq 20.
\end{align*}
\]  
(3.1)

The second problem (Problem 2) have a trigonometric objective function with linear and quadratic constraints:

\[
\begin{align*}
\min \quad & f(x_1, x_2) = -10 \cos(x_1 x_2) + x_1 x_2 / 10 + 10(x_1 + x_2) \sin(x_1 + x_2) \\
\text{s.t.} \quad & x_1 - x_2 \geq 0.5 \\
& x_1 x_2 \leq 15 \\
& 0 \leq x_1 \leq 1.5 \\
& -1 \leq x_2 \leq 1.
\end{align*}
\]  
(3.2)

The third problem (Problem 3) is to optimize a trigonometric objective function with quadratic constraints:
Figure 8. (a–c) Plots of the objective functions.

\[
\begin{align*}
\text{min} & \quad f(x_1, x_2) = \frac{\sin^3(2x_1) \sin(2x_2)}{x_1^3(x_1 + x_2)} \\
\text{s.t.} & \quad x_1^2 - x_2 + 1 \leq 0 \\
& \quad (x_1 - 2)^2 - x_2 + 1 \leq 0 \\
& \quad 0.1 \leq x_1 \leq 1 \\
& \quad 0.1 \leq x_2 \leq 7.
\end{align*}
\]
The plots of the three objective functions are shown in figure 8. There are zero, one and two local minima in the defined field for Problem 1, 2 and 3, respectively. The results of the NOM are shown in table 3. The results are compared with those of other optimization algorithms in table 3. All the investigated optimization algorithms provide very identical results. The running time is also shown in table 3. The NN objective function is trained before optimization and is the same for each method. The training time of the NN objective function is not included in the running time shown in table 3. Nelder Mead (a direct search method) uses the shortest computational time. The NOM uses less computational time than Genetic Algorithm, Differential Evolution and Particle Swarm Optimization (heuristics algorithms).

We tested NOM on objective functions with deeper NNs for the numerical examples. The number of hidden layers increases from one to five. It is found that NOM can still find the optimal solutions. The optimization time for each training NOM does not increase with the increase of depth of the NN objective function since the trainable parameters are limited to the neurons between the starting point layer and the input layer. However, it is expected that the objective functions become more complex and many local minima exist with deeper NNs, which is more difficult for global optimization based on gradient-based techniques. Thus, the validity of the proposed NOM for deeper NNs is warned and additional investigations are needed for deeper and complex topology of NNs. This is also suggested as future work based on the current investigation.

Although NOM is a gradient-based method, the gradients need not be calculated manually. By properly designing the architecture, the NOM uses the built-in automatic differentiation technique to calculate gradients. We compare NOM with a direct search method (i.e. Nelder Mead) and heuristics algorithms (i.e. Genetic Algorithm, Differential Evolution and Particle Swarm Optimization) since those methods need no manual calculation of gradients either. In the paper, the Adam optimizer is used to train the NOM. It is noted that other optimizers can also be used, e.g. Quasi-Newton’s methods. The optimizer selection is beyond this paper’s scope and is considered as users’ choice.

Table 3. Result comparison between NOM and other optimization algorithms.

| Problem | NOM | Genetic Algorithm | Differential Evolution | Particle Swarm Optimization | Neural Optimization Machine |
|---------|-----|-------------------|------------------------|----------------------------|-----------------------------|
| Problem 1 |     |                   |                        |                            |                             |
| $x_1$   | 13.660 | 13.660           | 13.660                  | 13.694                     | 13.680                      |
| $x_2$   | 0.000  | 0.000             | 0.000                   | 0.000                      | 0.001                       |
| $f(x_1, x_2)$ | -7.950 | -7.950            | -7.950                  | -7.949                     | -7.948                      |
| Time (s) | 3.2   | 55.2              | 137.4                   | 36.0                       | 25.7                        |
| Problem 2 |     |                   |                        |                            |                             |
| $x_1$   | 0.258  | 0.257             | 0.256                   | 0.256                      | 0.257                       |
| $x_2$   | -0.241 | -0.242            | -0.243                  | -0.243                     | -0.243                      |
| $f(x_1, x_2)$ | -9.983 | -9.984            | -9.984                  | -9.984                     | -9.984                      |
| Time (s) | 3.0   | 54.6              | 139.7                   | 35.3                       | 20.0                        |
| Problem 3 |     |                   |                        |                            |                             |
| $x_1$   | 0.100  | 0.100             | 0.100                   | 0.100                      | 0.101                       |
| $x_2$   | 5.463  | 5.464             | 5.464                   | 5.464                      | 5.464                       |
| $f(x_1, x_2)$ | -1.406 | -1.406            | -1.406                  | -1.406                     | -1.405                      |
| Time (s) | 3.0   | 57.5              | 143.5                   | 34.8                       | 21.8                        |
Multi-objective optimization problem

NOM has been validated for optimizing problems involving a single NN objective function. This section illustrates how NOM can be extended to other types of optimization problems, i.e. multi-objective optimization. Non-dominated Sorting Genetic Algorithm (NSGA-II) is used to validate NOM results. The NOM is tested for multi-objective optimization following the methodology presented in §2c. Pareto optimal solutions are obtained by implementing NOM multiple times. For each implementation, a value for the upper bound $U^{(i)}$ for objective $f_i(X)$ is adopted. By varying $U^{(i)}$, multiple Pareto optimal solutions can be obtained.

The multi-objective optimization problem is shown in equations (3.4). This is a problem with two quadratic objective functions and linear constraints. The plots of the objective functions are shown in figure 9.

$$\begin{array}{l}
\text{min} \\
\quad f_1 = (x_1 - 3)^2 + (x_2 - 7)^2 \\
\quad f_2 = (x_1 - 9)^2 + (x_2 - 8)^2 \\
\text{s.t.} \\
\quad 70 - 4x_2 - 8x_1 \leq 0 \\
\quad -2.5x_2 + 3x_1 \leq 0 \\
\quad -6.8 + x_1 \leq 0 \\
\quad 0 \leq x_1 \leq 10 \\
\quad 5 \leq x_2 \leq 15. \\
\end{array}$$

Figure 9. Plots of the objective functions of the multi-objective optimization problem.
The architecture shown in figure 7 is used for this problem. According to the bounded objective function method set-up in equation (2.14), \( f_1 \) is used as the single objective function, and \( f_2 \) is used as one of the constraints. The optimal solution \( X_2^* \) of \( f_2 \) is \( x_1 = 6.784 \) and \( x_2 = 8.309 \). The outcomes of \( f_2 \) and \( f_1 \) at \( X_2^* \), \( f_2 (X_2^*) \), and \( f_1 (X_2^*) \), are 0.501 and 1.611, respectively. According to Expression (15), \( f_2 (X_2^*) \) and \( f_1 (X_2^*) \) are used as the lower and upper limits for varying the constraint of \( f_2 \), \( U^{(2)} \). The hyperparameters for training the NN objective functions and the NOM are the same as those shown in tables 1 and 2. The results are shown in figure 10. The red points are obtained using the NOM. The NSGA-II, a multi-objective optimization technique, is also applied to this problem. The results are shown in black points in figure 10. As seen from figure 10, the results obtained from both methods are almost identical.

This work uses the bounded objective function method to solve the multi-objective optimization, which requires optimum solutions for individual objectives. That is to illustrate how NOM can be applied to different optimization problems such as multi-objective optimization. Future effort can be integrating NOM with other algorithms for specific optimization problems.

4. Demonstration for AM process optimization

AM provides numerous benefits, including flexibility in complex product geometry and conservation of materials and energy, and has been widely applied in the mechanical field. However, the rough surface, internal pores and residual stress in AM are detrimental to the fatigue performance of structures or components. Those defects are highly influenced by AM processing parameters [19]. Thus, AM process optimization is critical to ensure fatigue performance and structural integrity. There are at least two major challenges for AM process optimization. First, the AM process–property relationship is very complex, and the pure physics-based model is very difficult to characterize the multi-scale and multi-physics interaction of the AM process. A pure data-driven model needs tremendous experimental data for training and may violate physics knowledge [20,34]. Thus, physics-guided learning can potentially leverage the benefits of both physics-based and data-driven models. Next, AM process optimization must be compatible with physics and equipment constraints (e.g. scanning speed limit and laser power range). In addition, significant uncertainties exist in the process-property mapping, especially for fatigue performance. Thus, constrained optimization under uncertainties is required for AM process design. Different methodologies can address both challenges and need separate
training and optimization formulation. The proposed method solves the two challenges in a seamless single framework. The key idea is to develop a novel NN architecture to model the probabilistic AM process–property relationship and optimize the process parameters within the neural architecture framework.

(a) Review of physics-guided learning for the AM process–property relationship

Ti-6Al-4V (Ti-64) is extensively used in AM [35–40]. Typical applications of Ti-64 include turbine blades and compressors. In those cases, components may have fatigue failure due to cyclic loadings [41,42]. Therefore, the fatigue performance of AM Ti-64 is essential for ensuring structural integrity [19]. The processing parameters influence the AM quality for fatigue performance. The key factors are in-processing parameters and post-processing parameters.

A NN approach was developed for the fatigue modelling for AM Ti-64 [19]. The model is named the Probabilistic Physics-guided Neural Network (PPgNN). That model can obtain probabilistic-stress-life (P-S-N) relationships [43] under different processing parameters. PPgNN can produce physically meaningful P-S-N curves satisfying the slope and curvature constraints. An illustration of the P-S-N curve is shown in figure 11. It shows the relation between fatigue stress (Sa) and logarithmic fatigue life (log(N)). The blue points are the experimental failure data, and the red triangles are the runout data (i.e. the experiment stops before specimen failure). The 95% confidence interval (CI, dashed curves) and mean S-N curve (solid curve) are obtained from PPgNN. Due to the physics guidance in the NN, the fitted P-S-N curves satisfy that the variance (or CI) of the fatigue life increases and the curvature of the mean S-N curve decreases as the stress decreases.

The PPgNN model is used as the NN objective function, as shown in the dashed line box in figure 12. The inputs of the PPgNN are fatigue parameters, AM in-processing parameters and AM post-processing parameters. Specifically, fatigue parameters are the stress amplitude $S$ and stress ratio $R$. Among all the AM in-processing parameters, scanning velocity $v$, laser power $P$, hatch distance $h$ and layer thickness $t$ are considered. Heat temperature $HT$ and heat time $Ht$ are the considered AM post-processing parameters. The outputs of the PPgNN are the statistics of the fatigue life, which are mean $\mu$ and standard deviation $\sigma$. The square neurons are designed for missing data problems. For a comprehensive description of the PPgNN, readers may refer to [19].
Figure 12. Neural Optimization Machine for the design of processing parameters in additive manufacturing using a physics-guided NN as the objective function.

(b) Design of processing parameters in AM

The developed NOM is flexible and is not restricted to specific NN architectures or activation functions. In this section, we applied the NOM in a physics-guided NN [44] to design processing parameters in AM to optimize the fatigue life of Ti-64.

The architecture of the NOM shown in figure 12 is built according to §2b(i) for constrained optimization. The loss function to be minimized by the NOM is

\[
\text{Loss(NOM)} = - (\mu - 1.96\sigma) + \text{outputs of constraint neurons},
\]

where \( \mu - 1.96\sigma \) is the 2.5% lower bound of the fatigue life [41]. In other words, this problem is to maximize the lower bound. The constraints are the ranges of the design variables, and are listed next to the corresponding constraint neurons. The hyperparameters are the same as in previous examples. The stress amplitude and ratio are fixed to 520 MPa and 0.1, respectively. The results of the NOM, together with other optimization algorithms, are shown in table 4. The NOM achieves a similar result of \( \log_{10} (\mu - 1.96\sigma) \) as Genetic Algorithm, Differential Evolution and Particle Swarm Optimization. The Nelder Mead provides a worse result in this case. This problem has more variables than the test examples shown in §3a. As a result, the computational cost increases for all the other optimization algorithms. However, the computational cost is not shown to increase for the NOM.
Figure 13. Fatigue life comparison using the AM parameter design obtained in this paper and from the literature.

Table 4. Results of different optimization algorithms for AM fatigue optimization.

|                | Nelder Mead | genetic algorithm | differential evolution | particle swarm optimization | neural optimization machine |
|----------------|-------------|-------------------|-------------------------|-----------------------------|----------------------------|
| $v$ (mm s$^{-1}$) | 1053.2      | 1067.2            | 1066.6                  | 1064.1                      | 1064.9                     |
| $P$ (W)        | 177.8       | 170.0             | 170.2                   | 170.0                       | 171.6                      |
| $h$ ($\mu$m)   | 86.3        | 90.2              | 88.9                    | 90.6                        | 90.8                       |
| $t$ ($\mu$m)   | 59.8        | 59.9              | 59.9                    | 59.9                        | 59.9                       |
| $HT$ (°C)      | 797.1       | 797.5             | 797.0                   | 797.2                       | 796.9                      |
| $H_t$ (h)      | 5.0         | 5.0               | 5.0                     | 5.0                         | 5.0                        |
| $\log_{10}(\mu - 1.96\sigma)$ | 4.458      | 4.462             | 4.462                   | 4.462                       | 4.462                      |
| time/second    | 32.2        | 105.3             | 261.6                   | 162.2                       | 21.6                       |

We collected the AM parameters adopted in the literature and compared the lower bound of fatigue life using the AM parameter design obtained in this work and from the literature (shown in table 5). The PPGNN model is used to calculate the fatigue life under 520 MPa stress amplitude and 0.1 stress ratio. The results of the optimized design in this work and the six designs in the literature are shown in figure 13. It is shown that the fatigue life can be significantly increased using the optimized AM parameters compared with those used in the literature.

5. Discussion

NOM is a gradient-based method and converges to local minima. NOM uses random initial weights and multiple starting points to find multiple local minima from which global minima can be potentially obtained. This section discusses the impact of initial NN parameters and starting points. As stated at the end of §2b(i), the NOM uses the strategy of random initial NN parameters from the starting point layer to input layer and multiple starting points to explore the global minimum. The impacts of these operations are discussed below.

During the training of the NOM, the NN parameters from the starting point layer to input layer are updated. This is achieved by computing the derivatives of the loss function of the NOM
Table 5. AM parameters used in the literature.

| design no. | v (mm s⁻¹) | P (W) | h (µm) | t (µm) | HT (°C) | Ht (h) | ref. |
|------------|-------------|-------|--------|--------|---------|--------|-----|
| 1          | 1250        | 170   | 100    | 30     | 650     | 3      | [36]|
| 2          | 1250        | 200   | 80     | 30     | 820     | 1.5    | [45]|
| 3          | 1200        | 280   | 140    | 30     | 704     | 1      | [46]|
| 4          | 1250        | 170   | 100    | 30     | 650     | 4      | [47]|
| 5          | 1000        | 400   | 160    | 50     | 700     | 1      | [48]|
| 6          | 1200        | 280   | 140    | 30     | 704     | 1      | [49]|

Table 6. Number of local minima obtained by the NOM.

|                | 1 starting point | training NOM once | training NOM multiple times | multiple starting points |
|----------------|------------------|-------------------|----------------------------|-------------------------|
| initial \(\mathbf{w}^{(s)}\) is one. | 1                | 1                 |                            | 2                       |
| initial \(\mathbf{w}^{(s)}\) is random. | 1                | 2                 |                            | 2                       |

with respect to NN parameters according to the backpropagation algorithm:

\[
\frac{d\text{Loss}}{db^{(s)}} = f_b(b^{(s)}, \mathbf{w}^{(s)}, X_0), \tag{5.1}
\]

and

\[
\frac{d\text{Loss}}{d\mathbf{w}^{(s)}} = f_w(b^{(s)}, \mathbf{w}^{(s)}, X_0), \tag{5.2}
\]

respectively, where \(\text{Loss}\) is the NOM loss function, \(\mathbf{w}^{(s)}\) and \(b^{(s)}\) are the loss function from the starting point layer to input layer, respectively, and \(X_0\) is the training data as well as the starting point of the NOM. Equations (5.1) and (5.2) show that the derivatives are the functions of the \(\mathbf{w}^{(s)}\) and \(b^{(s)}\) at the previous step and \(X_0\). If the initial \(\mathbf{w}^{(s)}\) and \(b^{(s)}\) are fixed to be zero and one, respectively, the calculations of the derivatives in equations (5.1) and (5.2) at the first step are only the function of the starting point. In this way, the NOM will converge to the nearest local minimum from the starting point due to the steepest gradient descent algorithm. As stated in §2b(i), before training the NOM, random initial values are assigned to \(\mathbf{w}^{(s)}\). This introduces randomness in the first step of the gradient calculation. As the following updates of the weights and biases are functions of the results from the previous steps, the initial random weights are assigned to the NOM to find more local minima. Also, multiple starting points are used to train the NOM. The randomness of the initial weights and multiple starting points help the NOM converge to multiple local minima, which can obtain the global minimum.

The above discussion is demonstrated using the following optimization problem:

\[
\begin{align*}
\min \quad & f(x_1, x_2) = \frac{\sin^3(2x_1) \sin(2x_2)}{x_1^2(x_1+x_2)} \\
\text{s.t.} \quad & 0.1 \leq x_1 \leq 1 \\
& 0.1 \leq x_2 \leq 7.
\end{align*}
\tag{5.3}
\]

This problem is modified from Problem 3 by deleting the first and second constraints. As shown in figure 8c, two local minima exist in the defined domain, and one is the global minimum. This problem aims to test whether the NOM can obtain both local minima or just one of those.
The results of the number of local minima obtained by the NOM are shown in table 6. Six cases are investigated according to the number of starting points and whether initial $w^{(s)}$ are one or random. The initial $b^{(s)}$ is zero for all cases. The test results in table 6 show that the NOM can obtain both local minima with multiple starting points and random initial $w^{(s)}$. Therefore, we use random initial $w^{(s)}$ and five starting points for all the previous optimization problems. The starting points correspond to the five smallest objective function values determined by the grid search, as stated in §2b(i). This strategy finds multiple local minima, which is possible to find the global minima. However, the other optimization techniques used in the previous sections only try to find the global minima.

6. Conclusions

A novel NN approach, the NOM, is proposed for the constrained optimization of NN models. The objective functions for the NOM are NN models. The optimization process is conducted by the NN’s built-in backpropagation algorithm. The NOM solves optimization problems by extending the architecture of the NN objective function model. This is achieved by appropriately designing the NOM’s structure, activation and loss function. The NOM is very flexible and is extended for multi-objective optimization. The NOM is tested using numerical optimization problems for constrained optimization and multi-objective optimization. The results obtained from the NOM are compared with the Nelder Mead, Genetic Algorithm, Differential Evolution and Particle Swarm Optimization for single-objective optimization, and Non-dominated Sorting Genetic Algorithm (NSGA-II) for multi-objective optimization. The NOM is then applied to design processing parameters in AM. Based on the investigation of this paper, the following are the conclusions.

1. The NN objective function can have arbitrary architectures and activation functions.
2. The application of the NOM is not limited to specific optimization problems, e.g. linear and quadratic programming.
3. Multiple local minima can be found, which provides the potential for finding the global minimum.
4. The increase of the dimension of design variables does not increase the computational cost significantly for the NOM.

Data accessibility. Data are available via the following link: https://github.com/ymlasu/Neural-Optimization-Machine-NOM.

Authors’ contributions. J.C.: conceptualization, data curation, formal analysis, investigation, methodology, resources, software, validation, visualization, writing—original draft, writing—review and editing; Y.L.: conceptualization, data curation, formal analysis, funding acquisition, investigation, methodology, project administration, resources, software, supervision, validation, visualization, writing—review and editing.

Both authors gave final approval for publication and agreed to be held accountable for the work performed therein.

Conflict of interest declaration. We declare we have no competing interests.

Funding. The research in this paper was partially supported by funds from NASA University Leadership Initiative program (Contract no. NNX17AJ86A, Project Officer: Dr Anupa Bajwa, Principal Investigator: Dr Yongming Liu).

Acknowledgements. The support is gratefully acknowledged.

References

1. Effati S, Nazemi A. 2006 Neural network models and its application for solving linear and quadratic programming problems. *Appl. Math. Comput.* 172, 305–331. (doi:10.1016/j.amc.2005.02.005)
2. Forrester AJJ, Söbester A, Keane AJ. 2006 Optimization with missing data. *Proc. R. Soc. A* 462, 935–945. (doi:10.1098/rspa.2005.1608)
3. Xia Y, Feng G, Wang J. 2008 A novel recurrent neural network for solving nonlinear optimization problems with inequality constraints. *IEEE Trans. Neural Netw.* **19**, 1340–1353. (doi:10.1109/TNN.2008.2000273)

4. Zhang X-S. 2013 *Neural networks in optimization*. Berlin, Germany: Springer Science & Business Media.

5. Chen J, Gao Y, Liu Y. 2022 Multi-fidelity data aggregation using convolutional neural networks. *Comput. Methods Appl. Mech. Eng.* **391**, 114490. (doi:10.1016/j.cma.2021.114490)

6. Veerman MA, Pincus R, Stoffer R, van Leeuwen CM, Podareanu D, van Heerwaarden CC. 2021 Predicting atmospheric optical properties for radiative transfer computations using neural networks. *Phil. Trans. R. Soc. A* **379**, 20200095. (doi:10.1098/rsta.2020.0095)

7. Tank D, Hopfield J. 1986 Simple‘neural’optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit. *IEEE Trans. Circ. Syst.* **33**, 533–541. (doi:10.1109/TCS.1986.1085953)

8. Dhingra A, Rao S. 1992 A neural network based approach to mechanical design optimization. *Eng Optim.* **20**, 187–203. (doi:10.1080/03052159208941280)

9. Wu A, Tam PK-S. 1999 A neural network methodology of quadratic optimization. *Int. J. Neural Syst.* **9**, 87–93. (doi:10.1142/S0129065799000083)

10. Tagliarini GA, Christ JF, Page EW. 1991 Optimization using neural networks. *IEEE Trans. Comput.* **40**, 1347–1358. (doi:10.1109/12.106220)

11. Neelakantan T, Pandarikanthan N. 2000 Neural network-based simulation-optimization model for reservoir operation. *J. Water Resour. Plan. Manage.* **126**, 57–64. (doi:10.1061/(ASCE)0733-9496(2000)126:2(57))

12. Nascimento CAO, Giudici R, Guardani R. 2000 Neural network based approach for optimization of industrial chemical processes. *Comput. Chem. Eng.* **24**, 2303–2314. (doi:10.1016/S0098-1354(00)00587-1)

13. Darvishvand L, Kamkari B, Kowsary F. 2018 Optimal design approach for heating irregular-shaped objects in three-dimensional radiant furnaces using a hybrid genetic algorithm–artificial neural network method. *Eng. Optim.* **50**, 452–470. (doi:10.1080/0305215X.2017.1323889)

14. Jeon Y, Lee M, Choi JY. 2019 Neuro-Optimization: Learning Objective Functions Using Neural Networks. *arXiv preprint arXiv:1905.10079*.

15. Chandrasekhar A, Suresh K. 2021 TOuNN: Topology optimization using neural networks. *Struct. Multidiscip. Optim.* **63**, 1135–1149. (doi:10.1007/s00158-020-02748-4)

16. Villarrubia G, De Paz JF, Chamoso P, la Prieta FD. 2018 Artificial neural networks used in optimization problems. *Neurocomputing* **272**, 10–16. (doi:10.1016/j.neucom.2017.04.075)

17. Khalaj O, Jamshidi MB, Saebnoori E, Mašek B, Štadler C, Svoboda J. 2021 Hybrid machine learning techniques and computational mechanics: Estimating the dynamic behavior of oxide precipitation hardened steel. *IEEE Access* **9**, 156930–156946. (doi:10.1109/ACCESS.2021.3129454)

18. Khalaj O, Jamshidi M, Hassas P, Hosseininezhad M, Mašek B, Štadler C, Svoboda J. 2023 Metaverse and AI Digital Twinning of 42SiCr Steel Alloys. *Mathematics* **11**, 4. (doi:10.3390/math11010004)

19. Chen J, Liu Y. 2021 Fatigue property prediction of additively manufactured Ti-6Al-4V using probabilistic physics-guided learning. *Addit. Manuf.* **39**, 101876. (doi:10.1016/j.addma.2021.101876)

20. Chen J, Liu Y. 2020 Probabilistic physics-guided machine learning for fatigue data analysis. *Expert Syst. Appl.* **168**, 114316. (doi:10.1016/j.eswa.2020.114316)

21. Laddach K, Łangowski R, Rutkowski TA, Puchalski B. 2022 An automatic selection of optimal recurrent neural network architecture for processes dynamics modelling purposes. *Appl Soft Comput.* **116**, 108375. (doi:10.1016/j.asoc.2021.108375)

22. Chen J, Liu Y. 2021 Fatigue modeling using neural networks: a comprehensive review. *Fatigue Fract. Eng. Mater. Struct.* **45**, 945–979. (doi:10.1111/ffe.13640)

23. Akgun OC, Mei J. 2020 An energy efficient time-mode digit classification neural network implementation. *Phil. Trans. R. Soc. A* **378**, 20190163. (doi:10.1098/rsta.2019.0163)

24. Ketkar N. 2017 *Deep learning with Python*. Berlin, Germany: Springer.
25. Zhang Z, Li L, Lu J. 2021 Gradient-based fly immune visual recurrent neural network solving large-scale global optimization. *Neurocomputing* **454**, 238–253. (doi:10.1016/j.neucom.2021.05.002)

26. Rao SS. 2019 *Engineering optimization: theory and practice*. Hoboken, NJ: John Wiley & Sons.

27. Mack Y, Goel T, Shyy W, Haftka R. 2007 Surrogate model-based optimization framework: a case study in aerospace design. In *Evolutionary computation in dynamic and uncertain environments*, pp. 323–342. Berlin, Germany: Springer.

28. Sundaram A. 2022 Multiobjective multi verse optimization algorithm to solve dynamic economic emission dispatch problem with transmission loss prediction by an artificial neural network. *Appl. Soft Comput.* **124**, 109021. (doi:10.1016/j.asoc.2022.109021)

29. Price AR, Myerscough RJ, Voughtchov II, Marsh R, Cox SJ. 2009 Multi-objective optimization of GENIE Earth system models. *Phil. Trans. R. Soc. A* **367**, 2623–2633. (doi:10.1098/rsta.2009.0039)

30. Arora J. 2004 *Introduction to optimum design*. Cambridge, MA: Elsevier.

31. Carmichael D. 1980 Computation of Pareto optima in structural design. *Int. J. Numer. Methods Eng.* **15**, 925–929. (doi:10.1002/nme.1620150610)

32. Cohon JL. 2004 *Multiobjective programming and planning*. Courier Corporation.

33. Stadler W. 1988 *Multicriteria optimization in engineering and in the Sciences*. Springer Science & Business Media.

34. Chen J, Yu Y, Liu Y. 2022 Physics-guided mixture density networks for uncertainty quantification. *Reliab. Eng. Syst. Saf.* **228**, 108823. (doi:10.1016/j.ress.2022.108823)

35. Kumar P, Ramamurty U. 2020 Fatigue of additive manufactured Ti-6Al-4V, Part I: the effects of powder feedstock, manufacturing, and post-process conditions on the resulting microstructure and defects. *Int. J. Fatigue* **132**, 105358. (doi:10.1016/j.ijfatigue.2019.105358)

36. Wycisk E, Emmelmann C, Siddique S, Walther F. 2013 High cycle fatigue (HCF) performance of Ti-6Al-4V alloy processed by selective laser melting. *Adv. Mater. Res.* **816**, 134–139. (doi:10.4028/www.scientific.net/AMR.816-817.134)

37. Pegues JW, Shao S, Shamseai N, Sanaei N, Fatemi A, Warner DH, Li P, Phan N. 2020 Fatigue of additive manufactured Ti-6Al-4V, Part I: the effects of powder feedstock, manufacturing, and post-process conditions on the resulting microstructure and defects. *Int. J. Fatigue* **132**, 105358. (doi:10.1016/j.ijfatigue.2019.105358)

38. Sandgren HR, Zhai Y, Lados DA, Shade PA, Schuren JC, Groeber MA, Kenesei P, Gavras A. 2016 Characterization of fatigue crack growth behavior in LENS fabricated Ti-6Al-4V using high-energy synchrotron x-ray microtomography. *Addit. Manuf.* **12**, 132–141. (doi:10.1016/j.addma.2016.09.002)

39. Sharma A, Chen J, Diewald E, Imanian A, Beuth J, Liu Y. 2021 Data-driven sensitivity analysis for static mechanical properties of additively manufactured Ti–6Al–4V. *ASME J. Risk Uncert Eng. Syst. B Mech. Eng.* **8**, 011108. (doi:10.1115/1.4051799)

40. Chen J, Meng C, Gao Y, Liu Y. 2022 Multi-fidelity neural optimization machine for Digital Twins. *Struct. Multidiscip. Optim.* **65**, 340. (doi:10.1007/s00158-022-03443-2)

41. Chen J, Liu Y. 2020 Uncertainty quantification of fatigue properties with sparse data using hierarchical Bayesian model. In *AIAA Scitech 2020 Forum*, pp. 0680. (doi:10.2514/6.2020-0680)

42. Günther J, Krewerth D, Lippmann T, Leuders S, Tröster T, Weidner A, Biermann H, Niendorf T. 2017 Fatigue life of additively manufactured Ti–6Al–4V in the very high cycle fatigue regime. *Int. J. Fatigue* **94**, 236–245. (doi:10.1016/j.ijfatigue.2016.05.018)

43. Chen J, Liu S, Zhang W, Liu Y. 2020 Uncertainty quantification of fatigue SN curves with sparse data using hierarchical Bayesian data augmentation. *Int. J. Fatigue* **134**, 105511. (doi:10.1016/j.ijfatigue.2020.105511)

44. Kashinath K *et al.* 2021 Physics-informed machine learning: case studies for weather and climate modelling. *Phil. Trans. R. Soc. A* **379**, 20200093. (doi:10.1098/rsta.2020.0093)

45. Fousová M, Vojtěch D, Doubrova K, Daniel M, Lin C-F. 2018 Influence of inherent surface and internal defects on mechanical properties of additively manufactured Ti6Al4V alloy: comparison between selective laser melting and electron beam melting. *Materials* **11**, 537. (doi:10.3390/ma11040537)

46. Carrion PE, Soltani-Tehrani A, Thompson SM, Shamseai N. 2018 Effect of powder degradation on the fatigue behavior of additively manufactured as-built Ti-6Al-4V. In 2018 *International Solid Freeform Fabrication Symposium*. University of Texas at Austin.
47. Rafi HK, Starr TL, Stucker BE. 2013 A comparison of the tensile, fatigue, and fracture behavior of Ti–6Al–4V and 15-5 PH stainless steel parts made by selective laser melting. Int. J. Adv. Manuf. Technol. 69, 1299–1309. (doi:10.1007/s00170-013-5106-7)

48. Fatemi A, Molaei R, Sharifimehr S, Phan N, Shamsaei N. 2017 Multiaxial fatigue behavior of wrought and additive manufactured Ti-6Al-4V including surface finish effect. Int. J. Fatigue 100, 347–366. (doi:10.1016/j.ijfatigue.2017.03.044)

49. Carrion PE, Soltani-Tehrani A, Phan N, Shamsaei N. 2019 Powder recycling effects on the tensile and fatigue behavior of additively manufactured Ti-6Al-4V Parts. JOM 71, 963–973. (doi:10.1007/s11837-018-3248-7)