Computational Mechanics
https://doi.org/10.1007/s00466-022-02216-4

ORIGINAL PAPER

A note on rapid genetic calibration of artificial neural networks

T. I. Zohdi[1]

Received: 19 June 2022 / Accepted: 15 July 2022
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2022

Abstract
Artificial Neural Nets (ANN) have received huge attention in the scientific community over the last decade and are based on layered input-output type frameworks that are essentially adaptive nonlinear regressions of the form $\mathcal{O} = \mathcal{B}(\boldsymbol{I}, \boldsymbol{w})$, where $\mathcal{O}$ is a desired output and $\mathcal{B}$ is the ANN comprised of (1) **synapses**, which multiply inputs $(I_1, I_2, \ldots, I_M)$ by weights $(w_1, w_2, \ldots, N)$ that represent the input relevance to the desired output, (2) **neurons**, which aggregate outputs from all incoming synapses and apply activation functions to process the data and (3) **training**, which calibrates the weights to match a desired overall output. A primary issue with ANN is the calibration of the synapse weights. This calibration can be cast as a nonconvex optimization problem, whereby the cost/error function represents the normed difference between observed data and the output of the ANN for a selected set of weights. The objective is to select a set of weight which minimizes the cost/error. One family of methods that are extremely well-suited for this process are genetic-based machine-learning algorithms. The goal of this short communication is to illustrate this process on a clear model problem.

Keywords Neural-net · Calibration · Genetic algorithm

1 Introduction

Artificial Neural Nets (ANN) are based on layered input-output type frameworks that are essentially adaptive nonlinear regressions of the form

$$\mathcal{O} = \mathcal{B}(I_1, I_2, \ldots, I_M, w_1, w_2, \ldots, w_N), \tag{1.1}$$

where $\mathcal{O}$ is a desired output and $\mathcal{B}$ is the ANN comprised of:

- **Synapses**, which multiply inputs $(I_1, I_2, \ldots, I_M)$ by weights $(w_1, w_2, \ldots, w_N)$ that represent the input relevance to the desired output,
- **Neurons**, which aggregate outputs from all incoming synapses and apply activation functions to process the data and
- **Training**, which calibrates the weights to match a desired overall output.

For example, Fig. 1 illustrates a detailed ANN comprised of (1) Five layers (one input layer and four hidden layers) (2) 35 activation neurons (3+5+7+9+11) and (3) 223 weighted synapses. *The primary issue with ANNs is the calibration or "training" of the synapse weights.* The key components of an ANN can be summarized as follows (which is centered around training):

- STEP 1: Guess a set of trial weights (trials, $J = 1, 2, \ldots$), given by the vector $\boldsymbol{w}^{J=1}$, for the synapse weights and insert into the ANN (detailed construction shown shortly)

$$\mathcal{B}(\boldsymbol{I}, \boldsymbol{w}^J) = \mathcal{O}^J, \tag{1.2}$$

  which produces an overall trial output (Jth trial).
- STEP 2: Compute the error for the Jth trial

$$\mathcal{E}^J \stackrel{\text{def}}{=} ||\mathcal{O}^{desired} - \mathcal{O}^J||, \tag{1.3}$$

  where $\mathcal{O}^{desired}$ is the desired output, which could come from experimental/field data or results from a complex computational model of a system, where a reduced complexity ANN may be useful to represent the system.
- STEP 3: The minimization of the error by adjusting the weights, for the next trial $(J + 1)$:

$$\boldsymbol{w}^{J+1} = \boldsymbol{w}^J + \Delta \boldsymbol{w}^J \tag{1.4}$$

✉ T. I. Zohdi
zohdi@berkeley.edu

[1] Department of Mechanical Engineering, University of California, Berkeley, CA 94720-1740, USA

Published online: 09 August 2022

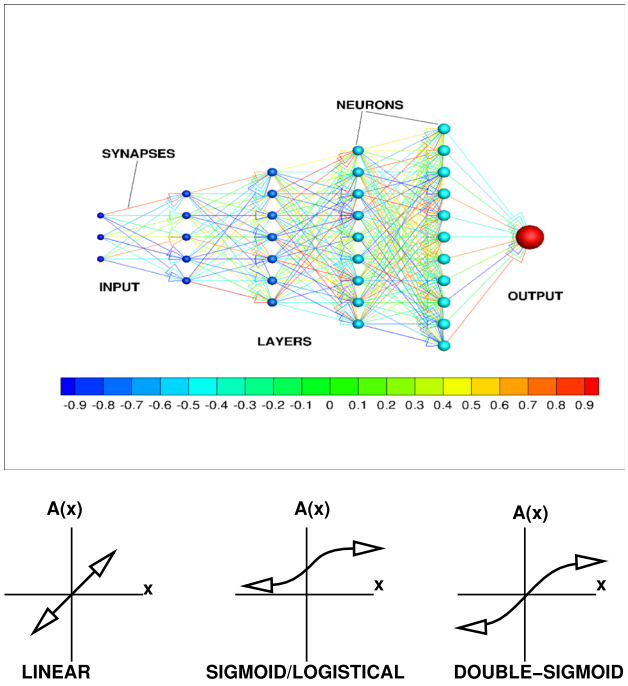

Fig. 1 Top: An ANN comprised of (1) Five layers (one input layer and four hidden layers) (2) 35 activation neurons (3+5+7+9+11) and (3) 223 weighted synapses. The color-coding represents the value of the synapse weights, in this case scaled by a factor of 10 (dark blue=-10 and bright red=+10). Bottom: Various neuron activation functions: (1) Linear (2) Sigmoid and (3) Double Sigmoid

- STEP 4: Repeat Steps 1-3 (trials $J = 1, 2, \ldots$) until the best set of weights are found to minimize the error.

The determination of the synapse weights can be cast as a nonconvex optimization problem, whereby the cost/error function represents the normed difference between observed data and the output of the ANN for a selected set of weights. The objective is to select a set of weights which minimizes the cost/error. One family of methods that are extremely well-suited for this process are genetic-based machine-learning algorithms. The goal of this short communication is to illustrate this process.

2 Construction of an ANN

The specifics of ANN construction are as follows:

- **Step 1:** Assign initial starting "guessed" weights to the synapses; this is the number of unknowns to be updated/optimized: $\boldsymbol{w}^J$, for trials $J = 1, 2, \ldots$
- **Step 2:** Collect (sum) the input (I_k, $k = 1, \ldots,$ *synapses*, Fig. 1) from each synapse input to each neuron:
- Neuron 1: $I_1 w_{1\to N_1} + I_2 w_{2\to N_1} + \ldots = S_{N_1}$, Neuron 2: $I_1 w_{1\to N_2} + I_2 w_{2\to N_2} + \ldots = S_{N_2}$, etc.
- **Step 3:** For each neuron, apply an activation function (Fig. 1) to process input: Neuron 1: $A_{N_1}(S_{N_1})$,Neuron 2: $A_{N_2}(S_{N_2})$, etc.
- Linear activation: $A(x) = x$ which provides proportional feedback
- Sigmoid/logistical activation: $A(x) = \frac{1}{1+e^{-x}}$, reinforces input for $x \to \infty$ and deletes input for $x \to -\infty$
- Double-sigmoid activation: $A(x) = \frac{1-e^{-x}}{1+e^{-x}}$, reinforces input for $x \to \infty$ and negates input for $x \to -\infty$
- **Step 4:** The output function sums contributions from the last layer:

$$\mathcal{O}(\Sigma_{i=1}^{N_{last}} w_{N_i\to O} A_{N_i}(S_{N_i})) \stackrel{\text{def}}{=} \mathcal{O}^J. \tag{2.1}$$

- **Step 5:** Compute the error $\mathcal{E}^J = ||\mathcal{O}^{Desired} - \mathcal{O}^J||$, in the appropriate problem specific norm, where $\mathcal{O}^{Desired}$ is observed data to be matched.
- **Step 6:** Repeat Steps 1-5 with an updated (improved) set of weights until a cost function $\Pi(\boldsymbol{\Lambda}^J) \stackrel{\text{def}}{=} \mathcal{E}^J$ is minimized by varying a design vector $\boldsymbol{\Lambda}^J \stackrel{\text{def}}{=} \{\Lambda_1^J, \Lambda_2^J, \ldots, \Lambda_N^J\} = \{w_1^J, w_2^J, \ldots, w_N^J\}$. This is the key step, referred to as *training or calibration*.

3 Genetic-based machine-learning optimization

The objective now is to minimize the cost function

$$\Pi(\Lambda_1^J, \ldots, \Lambda_N^J) \stackrel{\text{def}}{=} \mathcal{E}^J = ||\mathcal{O}^{Desired} - \mathcal{O}^J||, \tag{3.1}$$

by varying the design vector $\boldsymbol{\Lambda}^J \stackrel{\text{def}}{=} \{\Lambda_1^J, \Lambda_2^J, \Lambda_3^J, \ldots, \Lambda_N^J\} = \{w_1^J, w_2^J, w_3^J, \ldots, w_N^J\}$, for repeated trials $J = 1, 2, \ldots$. In order to cast the objective mathematically, we set the problem up as a genetic-based machine-learning algorithm, which is well-suited for nonconvex optimization. Following Zohdi [21–27], we formulate the objective as a cost function minimization problem that seeks system parameters that match a desired response, in this case a minimum of the neural network error in matching the given data, represented by $\Pi(\Lambda_1^J, \ldots \Lambda_N^J)$. We systematically minimize Eq. 3.1, $min_{\Lambda^J} \Pi$, by varying the design parameters: $\boldsymbol{\Lambda}^J \stackrel{\text{def}}{=} \{\Lambda_1^J, \Lambda_2^J, \Lambda_3^J, \ldots, \Lambda_N^J\}$. The system parameter search is conducted within the constrained ranges of $\Lambda_1^{(-)} \le \Lambda_1^J \le \Lambda_1^{(+)}$, $\Lambda_2^{(-)} \le \Lambda_2^J \le \Lambda_2^{(+)}$, $\Lambda_3^{(-)} \le \Lambda_3^J \le \Lambda_3^{(+)}$, etc. These upper and lower limits are dictated by what is physically feasible.

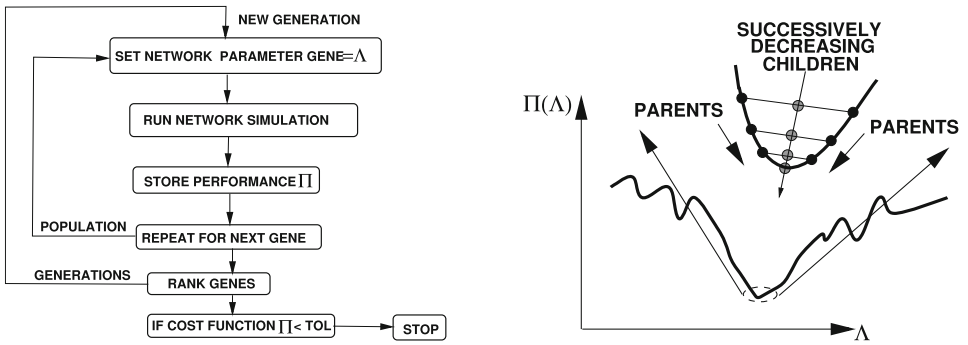

Fig. 2 The basic action of a genetic-based machine-learning algorithm (Zohdi [21–27])

3.1 Genetic-based machine-learning algorithm

Cost functions such as Π are nonconvex in design parameter space and often nonsmooth. Their minimization is usually difficult with direct application of gradient-based methods. This motivates nonderivative search methods, for example those found in machine-learning algorithms (MLAs). One of the most basic subsets of MLAs are so-called Genetic Algorithms (GAs). For a review of GAs, see the pioneering work of John Holland ([10,11]), as well as Goldberg [5], Davis [1], Onwubiko [18] and Goldberg and Deb [6]. A description of the algorithm will be described next, following Zohdi [21–27].

Remark 1 To be consistent with the genetic-algorithm terminology, we shall us the term "trial" interchangeably with "generation".

3.2 Algorithmic structure

The MLA/GA approach is extremely well-suited for nonconvex, nonsmooth, multicomponent, multistage systems and, broadly speaking, involves the following essential concepts (Fig. 2):

1. **POPULATION GENERATION:** Generate a parameter population $i = 1, 2, \ldots, S$ of genetic strings for the Jth generation: $\boldsymbol{\Lambda}^{J,i}$
2. **PERFORMANCE EVALUATION:** Compute performance of each genetic string: $\Pi(\boldsymbol{\Lambda}^{J,i})$
3. **RANK STRINGS:** Rank them $\boldsymbol{\Lambda}^{J,i}$, $i = 1, 2, \ldots, S$ from best to worst
4. **MATING PROCESS:** Mate pairs/produce offspring
5. **GENE ELIMINATION:** Eliminate poorly performing genetic strings
6. **POPULATION REGENERATION:** Repeat process with updated gene pool and new *random* genetic strings
7. **SOLUTION POST-PROCESSING:** Employ gradient-based methods afterwards in local "valleys"-*if smooth enough*

Remark 2 In order to make the upcoming presentation of the algorithm as clear as possible, we will write $\boldsymbol{\Lambda}^i = \boldsymbol{\Lambda}^{J,i}$, where it is assumed that this is the ith genetic string in the population at Jth trial/generation.

3.3 Specifics

Following Zohdi [21–27]. the algorithm is as follows:

- **STEP 1:** Randomly generate a population of S starting genetic strings, $\boldsymbol{\Lambda}^i$, $(i = 1, 2, \ldots, S)$, each containing N synapse weights:

$$\boldsymbol{\Lambda}^i \stackrel{\text{def}}{=} \left\{ \begin{array}{c} \Lambda_1^i \\ \Lambda_2^i \\ \Lambda_3^i \\ \ldots \\ \Lambda_N^i \end{array} \right\} \tag{3.2}$$

- **STEP 2:** Compute fitness of each string $\Pi(\boldsymbol{\Lambda}^i)$, (i=1,…, S)
- **STEP 3:** Rank genetic strings: $\boldsymbol{\Lambda}^i$, (i=1,…, S) from best to worst
- **STEP 4:** Mate nearest pairs and produce two offspring, (i=1,…, S):

$$\begin{aligned} \boldsymbol{\lambda}^i &\stackrel{\text{def}}{=} \boldsymbol{\Phi} \circ \boldsymbol{\Lambda}^i + (\mathbf{1} - \boldsymbol{\Phi}) \circ \boldsymbol{\Lambda}^{\mathbf{i+1}} \\ &\stackrel{\text{def}}{=} \left\{ \begin{array}{c} \phi_1 \Lambda_1^i \\ \phi_2 \Lambda_2^i \\ \phi_3 \Lambda_3^i \\ \ldots \\ \phi_N \Lambda_N^i \end{array} \right\} + \left\{ \begin{array}{c} (1-\phi_1)\Lambda_1^{i+1} \\ (1-\phi_2)\Lambda_2^{i+1} \\ (1-\phi_3)\Lambda_3^{i+1} \\ \ldots \\ (1-\phi_N)\Lambda_N^{i+1} \end{array} \right\} \end{aligned} \tag{3.3}$$

and

$$\boldsymbol{\lambda}^{i+1} \stackrel{\text{def}}{=} \boldsymbol{\Psi} \circ \boldsymbol{\Lambda}^{i} + (\mathbf{1} - \boldsymbol{\Psi}) \circ \boldsymbol{\Lambda}^{\mathbf{i+1}}$$
$$\stackrel{\text{def}}{=} \left\{ \begin{array}{c} \psi_1 \Lambda_1^i \\ \psi_2 \Lambda_2^i \\ \psi_3 \Lambda_3^i \\ \cdots \\ \psi_N \Lambda_N^i \end{array} \right\} + \left\{ \begin{array}{c} (1-\psi_1)\Lambda_1^{i+1} \\ (1-\psi_2)\Lambda_2^{i+1} \\ (1-\psi_3)\Lambda_3^{i+1} \\ \cdots \\ (1-\psi_N)\Lambda_N^{i+1} \end{array} \right\} \tag{3.4}$$

where for this operation, the ϕ_i and ψ_i are random numbers, such that $0 \leq \phi_i \leq 1$, $0 \leq \psi_i \leq 1$, which are different for each component of each genetic string

- **STEP 5:** Eliminate the bottom M strings and keep top K parents and their K offspring (K offspring+K parents+M=S)
- **STEP 6:** Repeat STEPS 1-5 with top gene pool (K offspring and K parents), plus M new, randomly generated, strings

Remark 3 If one selects the mating parameters $\phi's$ and $\psi's$ to be greater than one and/or less than zero, one can induce "mutations", i.e. characteristics that neither parent possesses. However, this is somewhat redundant with introduction of new random members of the population in the current algorithm. If one does not retain the parents in the algorithm above, it is possible that inferior performing offspring may replace superior parents. Thus, top parents should be kept for the next generation. Retained parents do not need to be reevaluated, thus also making the algorithm less computationally expensive. Numerous studies of the author (Zohdi [21–27]) have shown that the advantages of parent retention outweighs inbreeding, for sufficiently large population sizes.

Remark 4 One can refocus search around best performing parameter set every few generations, thus concentrating the computation effort around the most promising (optimal) areas of design space. This often has advantages for highly nonconvex problems.[1]

Remark 5 After application of such a global search algorithm, one can apply a gradient-based method around the best performing parameter set, if the objective function is sufficiently smooth in that region of the parameter space. In other words, if one has located a convex portion of the parameter space with a global genetic search, one can employ gradient-based procedures locally to minimize the objective function further, since they are generally much more efficient for convex optimization of smooth functions. An exhaustive review of these methods can be found in the texts of Luenberger [14] and Gill, Murray and Wright [4]. *However, refocussing usually makes this extra step unnecessary, since the search eventually concentrates the computational effort locally around the best parameter set beforehand.*

[1] We also note that this algorithm is extremely easy to parallelize.

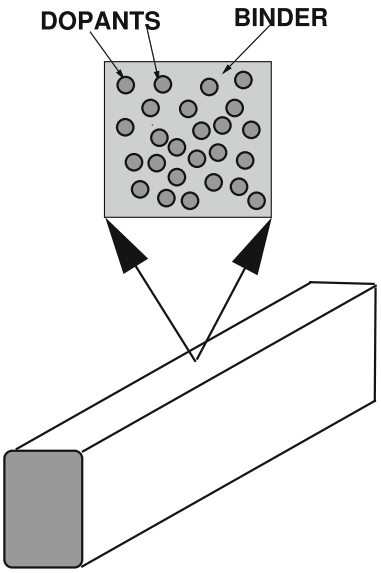

Fig. 3 A particle-functionalized material for structural applications

4 Example: Genetic training of an ANN with synthetic data material data

As an example, we consider a model problem in composite material design. The industrial use of particle-enhanced composite materials in structural applications is increasing (Fig. 3). Analysts are now afforded with many particle-matrix choices for possible material combinations. However, due to the nature of such applications, experiments to determine the appropriate combinations of particle and matrix materials are time-consuming and expensive, and it is advantageous to characterize such materials analytically and computationally, in order to reduce product development time and costs. In order to characterize the effective macroscale (structural) material response of such materials, a relation between averages,

$$\langle \boldsymbol{\sigma} \rangle_{\Omega} = \mathbb{E}^{*} : \langle \boldsymbol{\epsilon} \rangle_{\Omega}, \tag{4.1}$$

is sought, where

$$\langle \cdot \rangle_{\Omega} \stackrel{\text{def}}{=} \frac{1}{|\Omega|} \int_{\Omega} \cdot \, d\Omega \,, \tag{4.2}$$

and where, throughout the structure, the mechanical properties of microheterogeneous materials are characterized by a spatially variable elasticity tensor $\mathbb{E} = \mathbb{E}(\boldsymbol{x})$ and $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$ are the stress and strain tensor fields within a Representative Volume Element (RVE) of volume $|\Omega|$. The quantity $\mathbb{E}^{*}$ is known as the effective property. It is the elasticity tensor used in usual structural (macroscale) analyses. The internal fields, which are to be volumetrically averaged, can be computed by solving a series of boundary value problems with test load-

ings over an RVE using the Finite Element Method. However, this is extremely computationally intensive (Zohdi and Wriggers [28]) and oftentimes, faster approximate methods are sought. *Such approximations are important as a design tool.* We will concentrate on isotropic materials. In the case of isotropic overall responses, we may write

$$\left\langle \frac{tr\boldsymbol{\sigma}}{3} \right\rangle_{\Omega} = 3\kappa^* \left\langle \frac{tr\boldsymbol{\epsilon}}{3} \right\rangle_{\Omega} \tag{4.3}$$

and

$$\langle \boldsymbol{\sigma}' \rangle_{\Omega} = 2\mu^* \langle \boldsymbol{\epsilon}' \rangle_{\Omega}, \tag{4.4}$$

where κ^* and μ^* are the effective bulk and shear moduli, $\frac{tr\boldsymbol{\sigma}}{3}$ is the dilatational stress, $\frac{tr\boldsymbol{\epsilon}}{3}$ is the dilatational strain, $\boldsymbol{\sigma}'$ is the deviatoric stress and $\boldsymbol{\epsilon}'$ is the deviatoric strain. For an authoritative review of the general theory of random heterogeneous media see, for example, see Torquato [20] for general interdisciplinary discussions, Jikov et. al. [12] for more mathematical aspects, Hashin [7] or Mura [17] for solid-mechanics inclined accounts of the subject, for analyses of defect-laden, porous and cracked media, see Kachanov, Tsukrov and Shafiro [13] and for computational aspects see Ghosh [2], Ghosh and Dimiduk [3] and Zohdi and Wriggers [28].

4.1 Effective property estimates

The literature on methods to estimate the overall macroscopic properties of heterogeneous materials dates back at least to the 1800's by the pioneering works of Maxwell [15,16] and Lord Rayleigh [19], with an extremely important contribution being the Hashin-Shtrikman bounds during the 1960's (Hashin and Shtrikman [8,9], Hashin [7]). The Hashin-Shtrikman bounds are the tightest possible bounds on isotropic effective responses, generated from isotropic microstructures, where the volumetric data and phase contrasts of the constituents are the only data known. They are as follows, for the bulk modulus:

$$\begin{aligned} \kappa^{*,-} &\stackrel{\text{def}}{=} \kappa_1 + \frac{v_2}{\frac{1}{\kappa_2-\kappa_1} + \frac{3(1-v_2)}{3\kappa_1+4\mu_1}} \\ &\leq \kappa^* \leq \kappa_2 + \frac{1-v_2}{\frac{1}{\kappa_1-\kappa_2} + \frac{3v_2}{3\kappa_2+4\mu_2}} \stackrel{\text{def}}{=} \kappa^{*,+}, \end{aligned} \tag{4.5}$$

and for the shear modulus

$$\begin{aligned} \mu^{*,-} &\stackrel{\text{def}}{=} \mu_1 + \frac{v_2}{\frac{1}{\mu_2-\mu_1} + \frac{6(1-v_2)(\kappa_1+2\mu_1)}{5\mu_1(3\kappa_1+4\mu_1)}} \\ &\leq \mu^* \leq \mu_2 + \frac{(1-v_2)}{\frac{1}{\mu_1-\mu_2} + \frac{6v_2(\kappa_2+2\mu_2)}{5\mu_2(3\kappa_2+4\mu_2)}} \stackrel{\text{def}}{=} \mu^{*,+}, \end{aligned} \tag{4.6}$$

where κ_2 and κ_1 are the bulk moduli and μ_2 and μ_1 are the shear moduli of the respective phases ($\kappa_2 \geq \kappa_1$ and $\mu_2 \geq \mu_1$), and where v_2 is the second phase volume fraction. Phase 2 is the stiffer of the two constituents (which usually corresponds to the particles)[2]. One can form estimates for the effective properties by forming a convex combination of them, such as

$$\kappa^* \approx \eta\kappa^{*,+} + (1-\eta)\kappa^{*,-} \stackrel{\text{def}}{=} \kappa^{*,\eta} \tag{4.7}$$

and

$$\mu^* \approx \eta\mu^{*,+} + (1-\eta)\mu^{*,-} \stackrel{\text{def}}{=} \mu^{*,\eta} \tag{4.8}$$

where $0 \leq \eta \leq 1$ is a parameter such that:

- If $\eta = 0$ we have the lower bound,
- If $\eta = 1$ we have the upper bound and
- If $\eta = 1/2$ we have the average of the bounds.

Remark 6 If needed, one can post-process the effective bulk and shear modulus to obtain the effective Poisson ratio $\nu^* = \frac{3\kappa^*-2\mu^*}{2(3\kappa^*+\mu^*)}$ and the effective Young's modulus $E^* = 2\mu^*(1+\nu^*) = 3\kappa^*(1-2\nu^*)$.

4.2 Generation of a synthetic data set

As a test problem, we apply a neural net representation of the following effective property expression:

$$\Gamma(\mu_1, \mu_2, \kappa_1, \kappa_2, v_2) = c_1\mu^{*,\eta} + c_2\kappa^{*,\eta}, \tag{4.9}$$

where $c_1 = c_2 = 1$. Explicitly, we evaluated Γ at D=1000 random combinations of (μ_2, κ_2, v_2), with matrix material $(\mu_1, \kappa_1) = (0.1, 0.2)$ fixed (the averaging parameter was also fixed $\eta = 0.5$), generated between

- $1.75\mu_1 = \mu_2^- \leq \mu_2 \leq \mu_2^+ = 2.25\mu_1$,
- $1.75\kappa_1 = \kappa_2^- \leq \kappa_2 \leq \kappa_2^+ = 2.25\kappa_1$ and
- $0.25 = v_2^- \leq v_2 \leq v_2^+ = 0.75$.

4.3 Selected neural net configuration and cost function

As an example, we use an ANN (Fig. 1) comprised of (1) Five layers (one input layer and four hidden layers) (2) 35 activation neurons (3+5+7+9+11) and (3) 223 weighted synapses. Our objective is the use of genetic-based machine-learning to calibrate the 223 weights for the synapses. The cost function is

[2] Note that no geometric or statistical information is required for the bounds.

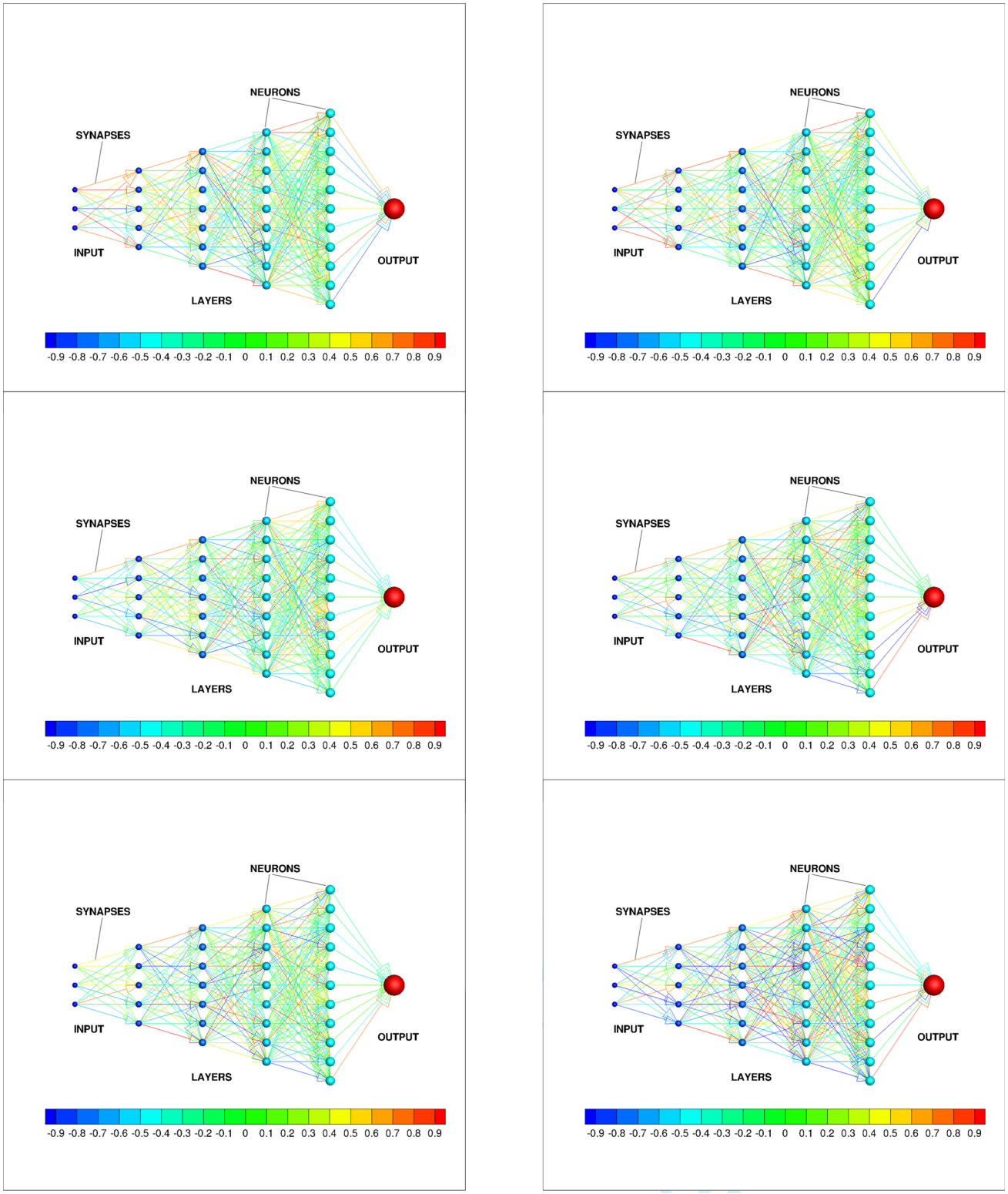

Fig. 4 The evolution of the best synapse weights, which stabilized after about 25 generations. From left to right and top to bottom at generations 1, 5, 10, 15, 20 and 25. The color-coding represents the value of the synapse weights, in this case scaled by a factor of 10 (dark blue=-10 and bright red=+10)

$$\Pi(\mathbf{\Lambda}^i) = \Sigma_{j=1}^{D=1000} |\Gamma(\mu_2^j, \kappa_2^j, v_2^j) - \mathcal{O}(\mathbf{\Lambda}^i, \mu_2^j, \kappa_2^j, v_2^j)|, \tag{4.10}$$

where $\mathbf{\Lambda}^i = \boldsymbol{w}^i$ is the design vector of synapse weights to be calibrated with the genetic-based machine-learning algorithm.

4.4 Algorithmic settings

In the upcoming example:

- Search parameter ranges: $\Lambda_i^- = -10 \leq \Lambda_i \leq \Lambda_i^+ = 10$ for the 223 synapse weights,
- Number of design variables: 223,
- Population size per generation: 24,
- Number of parents to keep in each generation: 6,
- Number of children created in each generation: 6,
- Number of completely new genes created in each generation: 12,
- Number of generations for re-adaptation around a new search interval: 20 and
- Number of generations: 50.

A double sigmoid function (Fig. 1) was used. Any advantage of one type of activation function over another, when training with genetic algorithms, is problem specific.

5 Discussion and summary

Figure 5 illustrates the data points used to train the algorithm. The sequence in Fig. 4 shows the changes in the weights at various stages of genetic calibration (generations 1, 5, 10, 15, 20 and 25). The weights were chosen to vary between -10 and +10, but of course there is nothing stopping choices outside of this range. This was done simply to allow for a clear presentation in the example. Figure 6 illustrates the results for the cost function for the best performing gene (*red*) as a function of successive generations, as well as the average performance cost function of the entire population of genes (*green*), and also shows the optimized synapse weights. The refocussing was implemented at generation 20, which reduced the best gene's cost function from approximately $\Pi \approx 0.55$ to $\Pi \approx 0.135$ over the course of 5 generations. The best gene's cost function stabilized at $\Pi \approx 0.135$ from a start of $\Pi \approx 6$; a reduction of error by a factor of 44.44. The approach is quite simple to implement and the entire 50 generation simulation, with 24 genes per evaluation (1,200 total designs) took under a second on a laptop, *making it ideal as a design tool.* This example was representative how this algorithm performed across a wide range of problems. The provided example, pertaining to the design of

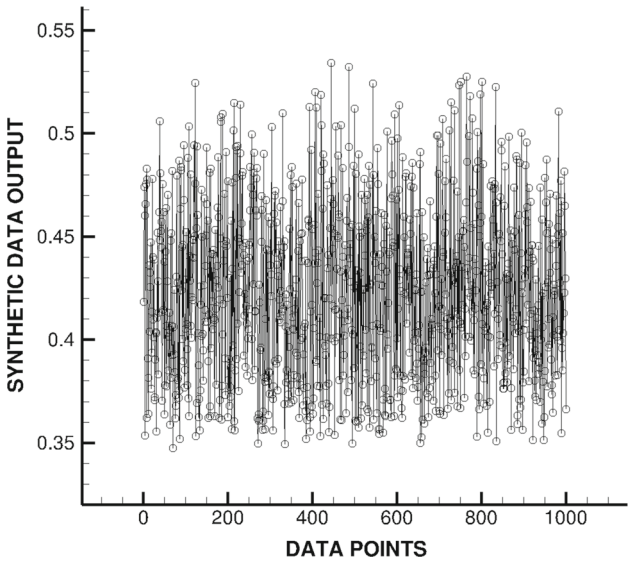

Fig. 5 One thousand synthetic data points used to train the Neural Net: $\mathcal{O}^{Desired}$

a heterogeneous isotropic material, was selected to be easily understandable by a wide audience. Of course, one could extend this type of example to more complex anisotropic materials, etc. However, the overall objective of this brief note is to illustrate how the genetic-calibration works for a neural network representation of any system, arising from solid mechanics, fluid mechanics, electromagnetics, etc. The physical problem used to generate the calibration data is somewhat irrelevant. Setting the number of data points to 1000 was arbitrary.

In summary, ANN are based on layered input-output type frameworks that are adaptive nonlinear regressions of the form $OUTPUT = \mathcal{B}(\boldsymbol{I}, \boldsymbol{w})$, comprised of (1) synapses and (2) neurons. A primary issue with ANN is the calibration of the synapse weights. This calibration can be cast as a nonconvex optimization problem, whereby the cost/error function represents the normed difference between observed data and the output of the ANN for a selected set of weights. The objective is to select a set of weight which minimizes the cost/error. The goal of this short communication was to illustrate a straightforward genetic-based calibration approach on a simple model problem. Genetic-based methods are extremely well-suited for this calibration process and quite easy to implement. Generally, it is vigorously debated as to what the balance should be between the number of data points used and the number of synapses (and hence synapse weights). It is the opinion of the author that this balance is *very* problem specific, due to the inherent nonconvexity of the underlying system to be optimized. However, the utility of the presented algorithm is that it is extremely fast and efficient, thus allowing one to easily explore adding/subtracting more synapse weights, varying layers, utilizing more data points, etc., in

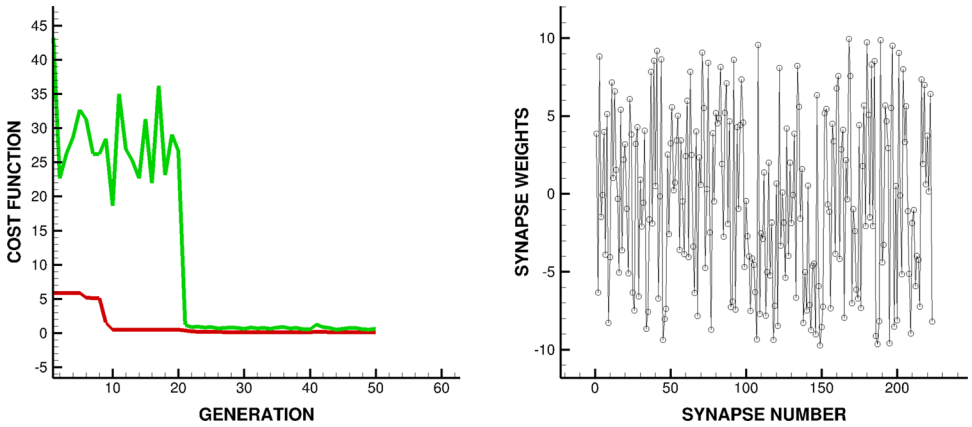

Fig. 6 Left: The cost/error function associated with the best performing gene (red) and the average of the population (green). Right: The values optimized (223) synapse weights that comprise the design vector

order to test various neural network combinations, with the goal being to provide the best possible accuracy for the specific problem at hand.

Acknowledgements This work has been partially supported by the UC Berkeley College of Engineering and the USDA AI Institute for Next Generation Food Systems (AIFS), USDA award number 2020-67021-32855.

References

1. Davis L (1991) Handbook of Genetic Algorithms. Thompson Computer Press
2. Ghosh S (2011) Micromechanical Analysis and Multi-Scale Modeling Using the Voronoi Cell Finite Element Method. CRC Press/Taylor & Francis
3. Ghosh S, Dimiduk D (2011) Computational Methods for Microstructure-Property Relations. Springer, NY
4. Gill P, Murray W, Wright M (1995) Practical optimization. Academic Press
5. Goldberg DE (1989) Genetic algorithms in search, optimization & machine learning. Addison-Wesley
6. Goldberg DE, Deb K (2000) Special issue on Genetic Algorithms. Computer Methods in Applied Mechanics & Engineering. 186(2–4):121–124
7. Hashin Z (1983) Analysis of composite materials: a survey. ASME Journal of Applied Mechanics. 50:481–505
8. Hashin Z, Shtrikman S (1962) On some variational principles in anisotropic and nonhomogeneous elasticity. J Mech Phys Solids 10:335–342
9. Hashin Z, Shtrikman S (1963) A variational approach to the theory of the elastic behaviour of multiphase materials. Journal of the Mechanics and Physics of Solids. 11:127–140
10. Holland JH (1975) Adaptation in natural & artificial systems. University of Michigan Press, Ann Arbor, Mich
11. Holland JH, Miller JH (1991) Artificial Adaptive Agents in Economic Theory (PDF). American Economic Review. 81 (2): 365-71. Archived from the original (PDF) on October 27, 2005
12. Jikov VV, Kozlov SM, Olenik OA (1994) Homogenization of differential operators and integral functionals. Springer-Verlag
13. Kachanov M, Tsukrov I, Shafiro B (1994) Effective moduli of solids with cavities of various shapes. Appl Mech Rev 47:S151–S174
14. Luenberger D (1974) Introduction to Linear & Nonlinear Programming. Addison-Wesley, Menlo Park
15. Maxwell JC (1867) On the dynamical theory of gases. Philos. Trans. Soc. London. 157:49
16. Maxwell JC (1873) A treatise on electricity and magnetism, 3rd edn. Clarendon Press, Oxford
17. Mura T (1993) Micromechanics of defects in solids, 2nd edn. Kluwer Academic Publishers
18. Onwubiko C (2000) Introduction to engineering design optimization. Prentice Hall
19. Rayleigh JW (1892) On the influence of obstacles arranged in rectangular order upon properties of a medium. Phil Mag 32:481–491
20. Torquato S (2002) Random Heterogeneous Materials: Microstructure & Macroscopic Properties. Springer-Verlag, New York
21. Zohdi TI (2018) Dynamic thermomechanical modeling and simulation of the design of rapid free-form 3D printing processes with evolutionary machine learning. Computer Methods in Applied Mechanics and Engineering Volume 331, 1 April 2018, Pages 343-362
22. Zohdi TI (2019) Electrodynamic machine-learning-enhanced fault-tolerance of robotic free-form printing of complex mixtures. Computational Mechanics. 63, pages 913-929 (2019)
23. Zohdi TI (2021) A Digital-Twin and Machine-learning Framework for the Design of Multiobjective Agrophotovoltaic Solar Farms. Comput Mech. https://doi.org/10.1007/s00466-021-02035-z
24. Zohdi TI (2021) A Digital-Twin and Machine-learning Framework for Ventilation System Optimization for Capturing Infectious Disease Respiratory Emissions. Archives of Computational Methods in Engineering. https://doi.org/10.1007/s11831-021-09609-3
25. Zohdi TI (2022) A digital-twin and machine-learning framework for precise heat and energy management of data-centers. Comput Mech. https://doi.org/10.1007/s00466-022-02152-3
26. Zohdi TI (2020) A machine-learning framework for rapid adaptive digital-twin based fire-propagation simulation in complex environments. Computer Methods Appl. Mech. Eng. 363:112907
27. Zohdi TI (2021) A digital twin framework for machine learning optimization of aerial fire fighting and pilot safety. Comput Methods Appl Mech Eng 373(1):113446
28. Zohdi TI, Wriggers P (2008) Introduction to computational micromechanics. Springer-Verlag
29. Zohdi TI, Monteiro PJM, Lamour V (2002) Extraction of elastic moduli from granular compacts. The International Journal of Fracture/Letters in Micromechanics. 115:L49–L54

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.