Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 327 (2017) 327–351
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Computational mechanics enhanced by deep learning

Atsuya Oishi[a,*], Genki Yagawa[b]

[a] *Graduate School of Technology, Industrial and Social Sciences, Tokushima University, 2-1 Minami-Johsanjima, Tokushima 770-8506, Japan*
[b] *Professor Emeritus, University of Tokyo and Toyo University, Setagaya, Tokyo 156-0044, Japan*

Available online 4 September 2017

Abstract

The present paper describes a method to enhance the capability of, or to broaden the scope of computational mechanics by using deep learning, which is one of the machine learning methods and is based on the artificial neural network. The method utilizes deep learning to extract rules inherent in a computational mechanics application, which usually are implicit and sometimes too complicated to grasp from the large amount of available data A new method of numerical quadrature for the FEM stiffness matrices is developed by using the proposed method, where a kind of optimized quadrature rule superior in accuracy to the standard Gauss–Legendre quadrature is obtained on the element-by-element basis. The detailed formulation of the proposed method is given with the sample application above, and an acceleration technique for the proposed method is discussed

Keywords: Deep learning; Artificial neural network; Numerical quadrature; Element stiffness matrix

1. Introduction

The core of computational mechanics has been to find approximate solutions for a variety of partial differential equations, which describe natural, physical, chemical phenomena mathematically, and the numerical methods that form the basis of computational mechanics, such as the finite element method (FEM), the finite difference method (FDM), the boundary element method (BEM), the particle methods and so on, have been employed to solve the simultaneous linear equations derived by discretization of the above partial differential equations with the elements, the lattices or the nodes.

The size of the simultaneous linear equations to be solved has been growing larger and larger. Therefore significant amount of research resources have been devoted to solve large scale systems of linear equations efficiently, and many solution techniques have been successfully developed to tackle large scale problems [1–3]. This trend in computational mechanics has been supported by the development of computers in the last decades.

Extraordinary progress of digital computers has big impacts in information science and technology including computational mechanics. Machine learning is among them [4,5]. It includes various methods and technologies, and

* Corresponding author.
E-mail address: aoishi@tokushima-u.ac.jp (A. Oishi).

http://dx.doi.org/10.1016/j.cma.2017.08.040
0045-7825/

many of them require a large amount of computation and would be useful only when faster computers are available. This is the reason machine learning has progressed with the advance of computers.

Machine learning has also received significant impacts from the unprecedented advance of networking technologies: the Internet that connects computers all over the world and the Internet of the Things (IoT) technology, where everything has its own Internet Protocol (IP) address and is connected to each other via Internet. These technologies have made it much easier to collect enormous amount of data. Deep learning emerged to analyze the big data to recognize implicit features out of them, to find inherent tendencies in them, and to classify them into several categories, and has become widely known after it won overwhelming victory over any other machine learning technique in benchmark test to recognize various images out of a great many image data collected via Internet [6]. Since then, deep learning has begun to be utilized in various fields [7].

It is well known that some machine learning methods, including the artificial neural network (ANN), which is the core technology of deep learning, based on the synaptic neuron model [8], the genetic algorithm (GA) based on the biological evolution model [9,10], the genetic programming (GP) [11,12], the evolutionary algorithm (EA) or the evolutionary strategy (ES) [10], the Monte Carlo method (MC) based on the probability theory [4], have been tested and utilized in a variety of computational mechanics fields. These methods have added soft or probabilistically ambiguous features to the rigid feature that is inherent in the conventional computational mechanics as the numerical solutions of partial differential equations, exploring new fields in computational mechanics that could not be tractable without them. As for their applications to computational mechanics, the GAs as well as the EAs have been employed in many applications as a tool for global search [13–19]. There are also several other methods applied to computational mechanics: the proper orthogonal decomposition (POD) and the singular value decomposition (SVD) [20,21], the support vector machine (SVM) [22], the multifidelity importance sampling [23], the particle swarm optimization (PSO) and the other similar algorithms inspired by the swarm intelligence [24,25], a set of methods based on the Bayesian statistics [26–28], and various machine learning methods [29–32].

Among others, the artificial neural networks have been successfully applied to some fields in the computational mechanics [33]. The artificial neural networks can be categorized into two groups based on their architectural structure: the feed forward neural networks and the mutually connected neural networks such as the Hopfield networks. The former, often called as the multilayer perceptrons, have much more applications to computational mechanics due to their capability of approximating nonlinear mappings: the estimations of the constitutive equations of materials, the nondestructive evaluations, the structural optimizations, and so on [34–39].

The neural networks have been applied to the modeling of materials equivalent to constitutive equations of materials of various types [40–45], applications related to the constitutive models with novel training algorithms and network structures [46–49], the optimization of material composition of composites [50], the modeling of hysteresis curves in various fields [51,52]. They have also been applied to various nondestructive evaluation methods [53–59] including the ultrasonic testing [54–56] and the electrical impedance tomography [57] and so on. There are also many applications to structural optimization problems, most of which use the neural networks to reduce time-consuming evaluations of individuals that occur in the iterative optimization loop driven by global search algorithm, such as the GA or the ES [60–67].

In most of the neural networks adopted in the applications described above, the data of learning patterns and the learning epochs are relatively small and simple, meaning that the mappings to be simulated by the neural networks in these applications are plain, or they are forced to be so by narrowing the scope of the solution space. The above mentioned simplifications of problems are due to the fact that neural networks often need very long training CPU time and show poor convergence if the problem to be solved is complex.

These bottlenecks of neural networks, however, have been resolved, since computer hardware and training algorithms for the neural networks have by far advanced as to be fast enough to train large neural networks. Accordingly, the neural networks including deep learning have become applicable to much wider class of problems that have been believed to be too complex to tackle.

Viewing the current situation above, we define first the framework of computational mechanics enhanced by deep learning and then show how it can be applied to develop the optimized numerical quadrature to calculate the FEM element matrices that are impossible to be challenged without deep learning.

As is well known, this problem is essential to the finite element method and requires a great amount of computation. Many researches, therefore, have been performed seeking faster or more efficient numerical quadrature: the speedup by using shortened quadrature rules [68], a fast quadrature using sum factorization [69], a fast quadrature implemented

on GPUs or multi-core processors [70–72], development of quadrature rules on arbitrary polygons based on symmetry and group theory [73], the reduced integration with hourglass control [74,75], the optimization among several quadrature rules [76].

As for the accuracy of the numerical quadrature in the element stiffness matrix calculation, it is well known that the elements of regular shapes, such as a square in the two-dimensional space and a cube in the three-dimensional space, can be accurately integrated with a few quadrature points, but those of irregular or distorted shape require more, often much more quadrature points to be accurately integrated. To deal with this issue, some numerical integration schemes based on the symbolic manipulation have been proposed [77,78]. Use of unsymmetrical elements is also proposed [79].

In the isogeometric analysis [80,81] and the NURBS-enhanced finite element method [82,83], where NURBS (Non Uniform Rational B-Spline) is adopted as the basis function, the numerical quadrature needs more computational load due to higher degrees and complexity of basis functions, and then researches for faster quadrature rules have been very active. Among them are a comparative study on numerical integration over two-dimensional NURBS-shaped domains [84], the reduced Bezier element quadrature rules for isogeometric analysis [85], an efficient matrix computation using sum factorization [86], efficient quadrature rules making good use of the higher degree of continuity of NURBS or B-Spline basis functions on the element boundary [85,87–91] and researches on the numerical integration of trimmed elements [92–94].

In this paper, we propose a new, general-purpose, optimized numerical quadrature enhanced by deep learning. In contrast to the fact that most of the quadrature rules described above depend on the special mathematical nature of the integrand and are only applicable to very limited cases, our method is general-purpose and is applicable not only to the finite element method and the isogeometric analysis, but also to various mesh free methods [95] including the element-free Galerkin method [96] and the free mesh method [97,98].

This paper consists of five sections as follows: Section 2 describes some preliminaries on machine learning, the neural networks, deep learning and then the proposed framework of the computational mechanics enhanced by deep learning. In Section 3, a new optimized numerical quadrature enhanced by deep learning is described with some numerical examples. Some issues related to the computational efficiency when using deep learning in applications are discussed in Section 4. Finally, Section 5 concludes the paper with summary.

2. Neural network, deep learning and computational mechanics

2.1. Machine learning and neural network

Machine learning includes a set of methods that manage to find some inherent rules and dependences, whether explicit or implicit, from a large amount of data not intuitively or by expert's insight but automatically, and then do the classification or the regression with them. There are many kinds of algorithms in the field of machine learning to be selected per target application, where the following five viewpoints should be taken into account:

(1) Selection of the appropriate algorithm considering input–output relationship, causal relationship, or classification of the application, which determines input data as well as output for machine learning.

(2) Cost of data preparation for machine learning.

(3) Computational cost required to acquire rules through training in machine learning.

(4) Computational cost required to apply the rules acquired in (3) to the classification or the regression for new data.

(5) Accuracy of the classification or the regression in (4).

Though the feedforward neural networks often require more training time in (3) than other machine learning methods, they usually need less estimation time in (4).

This fact, together with their versatility to the classification and the regression, makes them the first choice for practical applications.

2.2. Feed forward neural network

Basic structure of the feedforward neural network is shown in Fig. 1, which consists of layered, fundamental processing components, called units. A neural network of N layers consists of the first, the input layer, the second through $(N-1)$th, the hidden layers, and the Nth, the output layer.

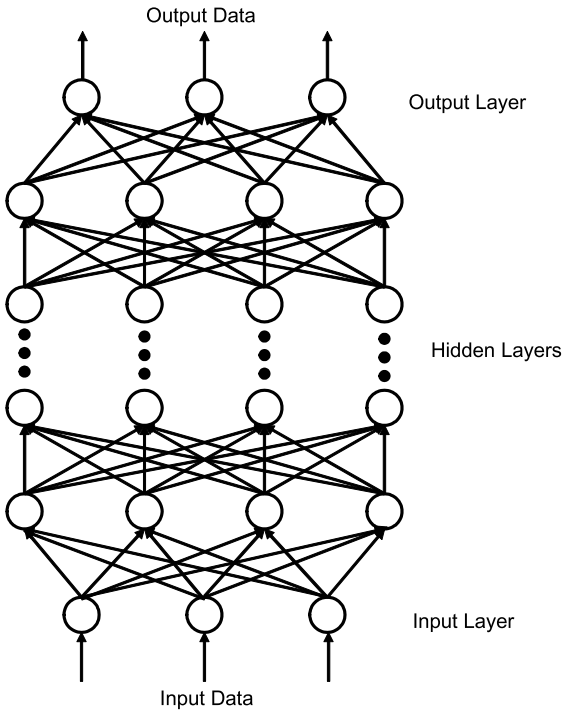


Fig. 1. Feed forward neural network consists of one input layer, one output layer and one or more hidden layers, each layer containing one or more units.

In a standard fully-connected feedforward neural network, every two units in the neighboring layers have a connection, to which a corresponding connection weight is attached, and no connections exist among units in the same layer as well as units in the non-neighboring layers.

Input data flow through a neural network via connections between units, starting from the input layer, and then through hidden layers, finally to the output layer, and the last layer outputs the processed data.

Assuming the number of units in the input layer and the output layer is n_I and n_O respectively, a feedforward neural network can be regarded as the mapping as follows:

$$NN \ : \ \mathbf{R}^{n_I} \rightarrow \mathbf{R}^{n_O}. \tag{1}$$

Here, while n_I and n_O can be determined just as the number of input and output data for the mapping NN, respectively, the number of units in the hidden layers as well as the number of hidden layers themselves can be any number and usually determined through trial-and-error procedure. It is proved that a feedforward neural network of more than three layers can simulate any non-linear continuous function with any precision desired [99,100].

A "unit" is a multiple-input–single-output processing device modeled after a nerve cell, called a neuron, taking output values from the units in the precedent layer as input and providing a single value as output. A schematic diagram of a unit is illustrated in Fig. 2. A unit in the hidden layers and the output layer takes as input the weighted sum of output values of the units in the precedent layer multiplied with corresponding connection weights and provides as output the value of activation function for the sum as follows:

$$O_j^p = f\left(U_j^p\right) = f\left(\sum_{i=1}^{n_{p-1}} w_{ji}^{p-1} \cdot O_i^{p-1} + \theta_j^p\right), \tag{2}$$

where O_i^{p-1} is the output of the ith unit in the $(p-1)$th layer, w_{ji}^{p-1} is the connection weight between the ith unit in the $(p-1)$th layer and the jth unit in the pth layer, θ_j^p is the bias of the jth unit in the pth layer, U_j^p is the input value to the activation function of the jth unit in the pth layer, O_i^p is the output value of the activation function of the jth unit in the pth layer, f () is the activation function, and n_{p-1} is the number of units in the $(p-1)$th layer.

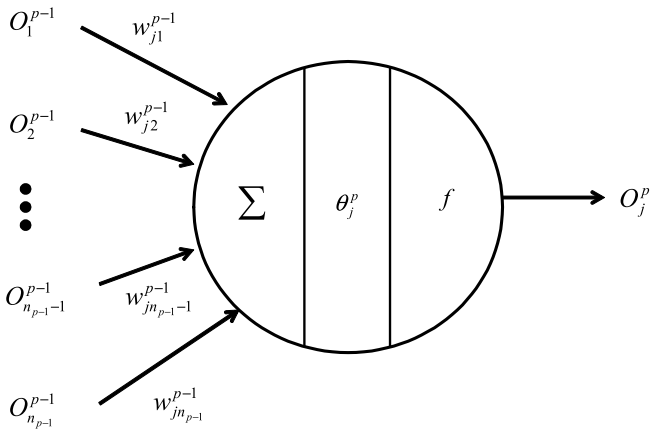


Fig. 2. At each layer, a unit sends single value O_j^p to all the units in the next layer after processing multiple data given by all the units in the precedent layer.

A nonlinear, non-decreasing function is used as an activation function. Among many functions, the sigmoid function and the rectified linear unit (ReLU) function is usually adopted as an activation function [101]. A linear function is also often adopted, especially for regression, as the activation function in the output layer. For the input layer, the identity function is usually adopted.

A feedforward neural network can construct on itself the mapping from the input data to the output data, i.e. the teacher signals, through iterative learning of those data pairs.

A pair of the input data and the corresponding output data, or the teacher signals, is called a training pattern, where the training is to construct, onto the neural network, the mapping relationship implicitly inherent in the large amount of training patterns.

The trained neural networks can, based on the mapping rules acquired through the training, provide estimated values for the new input data not included in the training data, which is called the generalization. As for the training algorithm, the error back propagation is usually adopted [102,103], where the sum of the squared error of all the output units for all training patterns is minimized:

$$E = \frac{1}{2}\sum_{q=1}^{nP}\sum_{j=1}^{n_N}\left({}^qO_j^N - {}^qT_j\right)^2, \tag{3}$$

where ${}^qO_j^N$ is the output of the jth unit in the output, Nth layer for the qth training pattern, ${}^qT_j$ is the teacher signal corresponding to the output of the jth unit in the output layer for the qth training pattern, and nP is the total number of training patterns.

When using the error, Eq. (3) as the loss function, the back propagation calculation is executed only once after accumulative forward propagations for all the training patterns. Instead of Eq. (3), another definition of the error is often adopted as

$$E = \frac{1}{2}\sum_{j=1}^{n_N}\left(O_j^N - T_j\right)^2 \tag{4}$$

where the error is defined for each training pattern.

When using this definition of error, the back propagation calculation is executed every time after a forward propagation of a single training pattern. The back propagation based on the direct minimization of the error Eq. (3) is called a batch training, and the one based on the minimization of the error Eq. (4) an on-line training [8]. Usually, the error defined in Eq. (3) gradually decreases epoch by epoch, and neural networks sometimes fit themselves to training patterns too strictly, which is called overfitting or overtraining. When it occurs, neural networks turn into a kind of look-up table and lose generalization capability of providing appropriate estimation for new input other than training patterns. To prevent the overfitting, one can prepare other patterns not included in training patterns, usually called test

patterns, monitor the error defined as Eq. (3) for the test patterns during training, and terminate the training before the error for test patterns starts to increase.

2.3. Deep learning

One usually tries to adopt neural networks with many numbers of hidden layers in order to gain better estimation capability for complex nonlinear problem, which often results in too long training time. This is partly because the update of the weights near the input layer of the neural network is insufficient, called as a vanishing gradient problem. Hinton et al. [104] have shown that a layer by layer training from the bottom to the top using the restricted Boltzmann machine is effective to solve this problem. This layer-wise training, called pretraining, sets the initial values of connection weights and biases, and then the network is trained as a whole using the back propagation algorithm. An autoencoder, a simple three-layer network with the same teacher signal as the input data, has been also shown to be effective for pretraining [105].

Supported also by recent advance of computers, we can now train large-scale neural networks with many hidden layers, which is called deep learning. According to LeCun et al. [106], training neural networks with more than three hidden layers is called deep learning. Better results being achieved with this method in the image classification problem [6], many interesting works have been reported, including the faster learning and the better accuracy of the training for neural networks: the dropout, which trains the ensemble consisting of subnetworks that can be formed by removing some units from an underlying base neural network [107], the maxout unit, which can learn the activation function itself [108], the AdaGrad algorithm, which adapts the learning rates by scaling them based on the historical values of themselves [109]. Since, in deep learning, a large amount of computation is commonly required for big training data, the accelerators such as GPUs are often employed with some libraries [110,111].

2.4. DL-Enhanced computational mechanics

In the computational mechanics, we first prepare a partial differential equation modeling a natural or physical phenomenon of our interest, and develop or select a method of discretization to solve the above equation numerically. Then, a matrix solver is selected, depending on the size and the computational nature of the equation to be solved. Throughout the process above, we select or develop rules to solve the problem.

It is noted here that the invention and the development of rules above have exclusively depended on man's insight. Careful observation on a natural or physical phenomenon and deep mathematical insight on it has led to a partial differential equation describing the phenomenon, and with plenty of data measured and some mathematical thought, we have developed a constitutive model of material.

In other words, to develop a new rule requires considerable efforts of experts in the related fields, and it is useful only when some assumptions and idealizations hold, reflecting the limits of man's ability of thinking.

Meanwhile, amazing advance of computers and networking technology has brought us the capability to deal with a large amount of data, and another way to find and develop a new rule; we find a rule "automatically" by using machine learning. Among many machine learning methods, deep learning has been found to be superior to any other methods in developing new rules. Based on the large amount of data, deep learning can find rules that are otherwise too complex to grasp. It is noted that a rule acquired through deep learning is kind of different from the conventional one given by mathematical equations and algorithms, which is rigid and explicit, reflecting the fact that it is derived through some idealizations.

A rule acquired through deep learning is, on the other hand, not represented in mathematical equations but in implicit mapping in which we cannot find any explicit rules. It has been indicated that deep learning can deal with much more complex relationship among many parameters than conventional rigid rules, which means that deep learning generates flexible and implicit rules, and that, only from a lot of input–output data given, deep learning can manage to find a mapping relationship among them and reconstruct it on itself.

The deep-learning (DL) enhanced computational mechanics consists of the following three phases:

(1) Data preparation phase: In any field or process in the computational mechanics, an input–output relationship is considered: the n-dimensional vector $\left\{x_1^p, x_2^p, x_3^p, \ldots, x_{n-1}^p, x_n^p\right\}$ as the input and the m-dimensional vector $\left\{y_1^p, y_2^p, y_3^p, \ldots, y_{m-1}^p, y_m^p\right\}$ as the output, where p means the pth data pairs out of a large set of data pairs and each component of the vector is of integer or real number type. It is noted a large amount of these data pairs is collected to

explore a mapping relationship over them, which can be efficiently collected employing the computational mechanics simulations.

(2) Training phase: Using the input–output data pairs collected in (1), deep learning starts the training to explore the mapping rules over them: a neural network is trained by taking the n-dimensional data $\left\{x_1^p, x_2^p, x_3^p, \ldots, x_{n-1}^p, x_n^p\right\}$ as the input to the network and the m-dimensional data $\left\{y_1^p, y_2^p, y_3^p, \ldots, y_{m-1}^p, y_m^p\right\}$ as the corresponding teacher signal. When the training converges, we have a multi-dimensional mapping f ();

$$f \ : \ \mathbf{R}^n \rightarrow \mathbf{R}^m. \tag{5}$$

In other words, we have the input–output relationship as follows;

$$\left(y_1^p, y_2^p, y_3^p, \ldots, y_{m-1}^p, y_m^p\right) = f\left(x_1^p, x_2^p, x_3^p, \ldots, x_{n-1}^p, x_n^p\right). \tag{6}$$

(3) Application phase: The input–output relationship given in (1) is replaced by the mapping achieved by deep learning: given input data, the corresponding output data are directly estimated by the mapping.

It sometimes occurs for the DL-enhanced computational mechanics that the neural network already trained is re-trained with the additional data pairs collected in the application phase or generated from new simulations to extend its applicability or to improve its accuracy.

3. Numerical quadrature enhanced by DL

3.1. Numerical quadrature

The FEM element matrices are usually calculated using numerical quadrature, where the integrated value of a function is approximated by the sum of the values of an integrand evaluated at several prescribed points multiplied by the corresponding weights. The first choice is the Gauss–Legendre quadrature, where an integrand $f\ (x)$ is numerically integrated in the range [-1, 1] as follows:

$$\int_{-1}^{1} f\ (x)\, dx \approx \sum_{i=1}^{n} f\ (x_i) \cdot H_i \tag{7}$$

where x_i is the ith integration point, H_i the weight corresponding to x_i and n the total number of integration points. Here, the coordinates of integration points and their corresponding weights are defined using the Legendre polynomials and the Lagrange polynomials. The former is defined as follows:

$$P_n\ (x) = \frac{d^n}{dx^n}\left(x^2 - 1\right)^n. \tag{8}$$

The following equation has n different real number of solutions $\{x_1, x_2, x_3, \ldots, x_{n-1}, x_n\}\ (x_i < x_{i+1})$ in the range of $(-1.0,\ 1.0)$, which are used as the coordinates of integration points in the Gauss–Legendre quadrature:

$$P_n\ (x) = 0. \tag{9}$$

The weights in the Gauss–Legendre quadrature are calculated using the Lagrange polynomial as follows:

$$L_i^{n-1}\ (x) = \frac{(x - x_1) \cdots (x - x_{i-1})\ (x - x_{i+1}) \cdots (x - x_n)}{(x_i - x_1) \cdots (x_i - x_{i-1})\ (x_i - x_{i+1}) \cdots (x_i - x_n)} \tag{10}$$

and

$$H_i = \int_{-1}^{1} L_i^{n-1}\ (x)\, dx \tag{11}$$

where $L_i^{n-1}\ (x)$ is the ith Lagrange polynomial of $(n-1)$th degree constructed from n solutions of Eq. (9) and H_i the weight at the ith integration point in the Gauss–Legendre quadrature. It is known that the Gauss–Legendre quadrature with n integration points is equivalent to integrate exactly the polynomial of $(2n-1)$th degree that approximates the integrand.

The Gauss–Legendre quadrature in one dimension is extended to that in two and three dimensions, respectively, as follows:

$$\int_{-1}^{1}\int_{-1}^{1} f(x,y)\,dxdy \approx \sum_{i=1}^{n}\sum_{j=1}^{m} f\left(x_i, y_j\right)\cdot H_{ij} \tag{12}$$

and

$$\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} f(x,y,z)\,dxdydz \approx \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l} f\left(x_i, y_j, z_k\right)\cdot H_{ijk} \tag{13}$$

where n, m and l are the numbers of integration points along x, y and z axes, respectively.

3.2. Element matrix

Using the finite element method, the discretization of the equilibrium equation of a static structural problem results in the following equation:

$$[K]\{u\} = \{f\} \tag{14}$$

where $\{u\}$ is the displacement vector, $[K]$ the global stiffness matrix, and $\{f\}$ the external force vector, where the global stiffness matrix is constructed by summing up all the element stiffness matrices as follows:

$$[K] = \sum_{e=1}^{n_e}\left[k^e\right] = \sum_{e=1}^{n_e}\int_{v^e}[B]^T[D][B]\,dv \tag{15}$$

where n_e is the total number of elements, $[k^e]$ the element stiffness matrix of the eth element, $[D]$ the stress–strain matrix and $[B]$ the strain–displacement matrix.

The integral $[k^e]$ defined in the real xyz space is transformed to that in the parametric $\xi\eta\zeta$ space of $[-1,\ 1]\times[-1,\ 1]\times[-1,\ 1]$ and then numerically integrated by the Gauss–Legendre quadrature as follows:

$$\left[k^e\right] \approx \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l}\left([B]^T[D][B]\cdot|J|\right)\Big|_{\substack{\xi=\xi_i\\ \eta=\eta_j\\ \zeta=\zeta_k}}\cdot H_{i,j,k} \tag{16}$$

where $|J|$ is the determinant of the Jacobi matrix, n, m and l the number of integration points along $\xi-$, $\eta-$ and ζ-axis respectively, and $H_{i,j,k}$ the weight of the integration at the quadrature point $\left(\xi_i, \eta_j, \zeta_k\right)$ defined by multiplying together all the weights along each axis. As is apparent from Eq. (16), the computational load required to perform the quadrature grows almost in proportion to the total number of integration points. Fig. 3 shows two typical 3D solid elements in the $\xi\eta\zeta$ space: an eight-node linear element (node 1 to 8) and a twenty-node quadratic serendipity one (node 1 to 20). An element stiffness matrix defined in Eq. (16) results in the size of 24 rows and 24 columns in the case of an eight-node linear element, 60 rows and 60 columns in the case of a twenty-node quadratic element.

Here, we will define a single measure for the quadrature related accuracy of element stiffness matrix, defined as follows:

$$Error = Error(q) = \frac{\sum_{i,j}\left|k_{i,j}^{q} - k_{i,j}^{q_{\max}}\right|}{\max_{i,j}\left|k_{i,j}^{q_{\max}}\right|}\quad , \tag{17}$$

where $k_{i,j}^{q}$ is the component at the ith row and the jth column of the element stiffness matrix obtained by numerical quadrature with q integration points, $q_{\max}$ the maximum of the numbers of integration points per axis to be used to compute the reference matrix, which is big enough for accurate approximation of the integrand. In this paper, we set $q_{\max}$ being 30; the reference matrix is obtained using the Gauss–Legendre quadrature with 30 integration points in total for a one-dimensional element, and 27,000 points for a three-dimensional element.

Tested is the accuracy of the element stiffness matrix obtained by using the Gauss–Legendre quadrature for some solid elements as shown in Fig. 4, where Fig. 4(a) illustrates an eight-node linear solid element of standard cubic

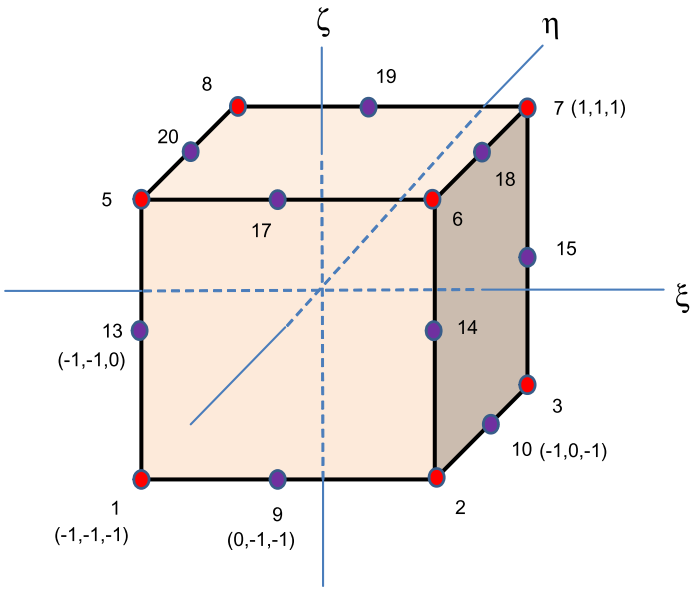


Fig. 3. 8 and 20 noded 3D linear and quadratic elements, respectively.

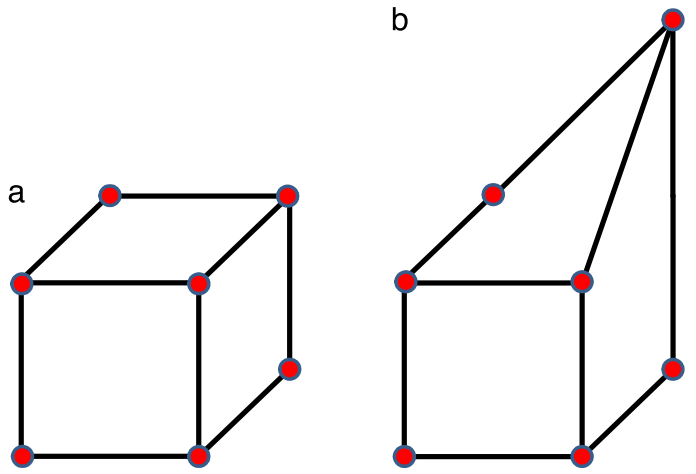


Fig. 4. Regular cubic and irregular distorted elements.

shape, and Fig. 4(b) that of distorted shape, where one of the edges is extended twofold and the two neighboring edges extended by the factor $\sqrt{2}$. By adding nodes at the middle of edges of the elements as shown in Fig. 4, two twenty-node quadratic elements are generated.

For these four kinds of elements, the convergence properties of the Gauss–Legendre quadrature are tested as shown in Fig. 5. The horizontal axis represents the number of integration points per axis, which is the same for all three axes, and the vertical axis shows the $Error$ defined in Eq. (17). In Fig. 5, "Linear-Regular and Quadratic-Regular" represent the linear and the quadratic elements of the shape illustrated in Fig. 4(a), respectively, and "Linear-Irregular and Quadratic-Irregular" those of the shape illustrated in Fig. 4(b). Though an element of the regular shape, whether it is linear or quadratic, shows fast convergence with just a few integration points to reach prescribed accuracy, that of distorted shape results in slower convergence requiring more integration points.

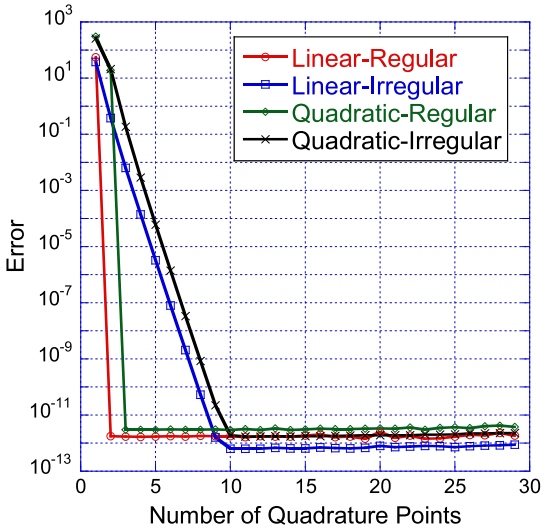


Fig. 5. Convergency of numerical quadrature in calculating stiffness matrices for linear and quadratic elements of regular and irregular shapes, respectively, where vertical axis represents the error defined in Eq. (17).

3.3. DL-enhanced numerical quadrature: basic concepts

It is well known that the Gauss–Legendre quadrature is an excellent and near optimal universal quadrature formula, which always uses the same coordinates and weights of the integration points. However, if these parameters are optimized case by case, a better quadrature formula specific to the integrand is expected. To make the concept clear, we show a simple integral, which is analytically integrated as follows:

$$\int_{-1}^{1} x^{10} dx = \left[\frac{1}{11}x^{11}\right]_{-1}^{1} = \frac{2}{11}. \tag{18}$$

Using the Gauss–Legendre quadrature formula with 2 integration points in the above, we have:

$$\int_{-1}^{1} x^{10} dx \approx 1.0 \cdot \left(\frac{-1}{\sqrt{3}}\right)^{10} + 1.0 \cdot \left(\frac{1}{\sqrt{3}}\right)^{10} = \frac{2}{243}. \tag{19}$$

In this case, the error obtained by the Gauss–Legendre quadrature is shown to be very large. If we try another case of both of the weights of integration in Eq. (19) being 243/11, then the integrand is shown to be exactly integrated using two integration points, suggesting that the value 243/11 is the optimal weight for the integrand in the above numerical quadrature using two integration points. Note that not two but six integration points give a correct answer in this case. In general, however, it is not easy to know how many integration points are required to have a solution within the prescribed range of error.

We propose here a new method considering that the optimization of the numerical quadrature can be realized with the following steps:

(A) The integrand is identified, which means that a unique set of parameters to identify the integrand is determined.

(B) For the integrand identified in (A), given the number of integration points, the numerical integration is performed using the optimized parameters, such as the coordinates and weights of the integration points.

(B') For the integrand identified in (A), the standard Gauss–Legendre quadrature of the integrand is performed using the appropriate number of integration points for the target accuracy given.

Noted that it is almost impossible to exactly identify an integrand out of so many function families, and to prepare the data base to be used in (B) or (B') that contains various parameters of numerical quadrature. However, if the target integrand is assumed to lie in smaller range of function families, the mapping of the integrand to a unique set of numbers in (A) as well as the construction of the data base used in (B) or (B') that contains optimized quadrature

parameters will become feasible, implying that the optimization of the numerical quadrature based on (A) and (B) or (B') will be possible.

Since an integrand seen in the integral of the FEM stiffness matrix consists mainly of the derivatives of the basis functions, which are usually shared among the elements of the whole continuum and of very limited varieties, it satisfies the condition (A) above.

As for the condition (B) or (B') above, too large amount of memory space is required to prepare parameters for each integrand, which varies depending on the coordinates of the nodes that construct the element. Then, we will not use the huge data base that contains all the information, but the mapping itself, i.e. the rule to determine the optimized parameters, where the selection of quadrature parameters for a given integrand is regarded as a mapping from the characteristics of the integrand, i.e. the set of numbers in (A), to the corresponding optimized parameters, i.e. the item in the data base to be used in (B) or (B'). With this mapping, we may easily get the optimized quadrature parameters for any integrand of the integral of the element stiffness matrix. Here, deep learning is employed to derive the mapping relationship from the characteristics of the integrand to the optimized parameters and to construct the mapping on the feedforward neural network.

3.4. DL-enhanced numerical quadrature: application to finite elements

In the integration of the FEM stiffness matrix, the integrand is usually constructed from the derivatives of the very limited variety of basis functions, identified by a small set of element parameters including the type of basis functions used and the coordinates of the nodes in the element, etc. Then, the optimized quadrature parameters for each integrand, such as the coordinates and the weights of the integration points, can be determined based on the corresponding element parameters. The minimum number of integration points required to integrate the integrand within the error of the prescribed range can also be determined based on the corresponding element parameters.

In this study, the following two types of optimizations are proposed:

(1) Optimization of the number of integration points in the Gauss–Legendre quadrature of the element stiffness matrix: the minimum number of integration points in the quadrature required for the prescribed range of error is estimated by deep learning. This optimization makes it possible to use more integration points only for elements which otherwise would produce relatively larger error than others in the numerical quadrature, and is equivalent to produce a global stiffness matrix accurate enough within the prescribed error using minimum number of integration points in total. Note that the standard Gauss–Legendre quadrature rule is used.

(2) Optimization of the parameters of the Gauss–Legendre quadrature for the element stiffness matrix: the optimized parameters of the quadrature, i.e. a set of parameters with which the quadrature produces the most accurate results, are estimated by deep learning. This optimization is equivalent to using different quadrature rules element by element, each of which is generated by deep learning and can do more accurate quadrature for the corresponding element than the standard Gauss–Legendre rule.

If we assume that the weights alone are adopted as the quadrature parameters to be optimized for simplicity, Eq. (16) is rewritten by using new parameters $\left\{w_{i,j,k}\right\}$ as

$$\left[k^e\right] \approx \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l}\left([B]^T[D][B]\cdot|J|\right)\Big|_{\substack{\xi=\xi_i\\ \eta=\eta_j\\ \zeta=\zeta_k}} \cdot H_{i,j,k}\cdot w_{i,j,k} \tag{20}$$

where $w_{i,j,k}$ is the correction factor to the corresponding standard weight of integration $H_{i,j,k}$ in the Gauss–Legendre quadrature rule. To identify the optimized weights is equivalent to determination of the set of correction factors $\left\{w_{i,j,k}\right\}$ to produce the most accurate results.

3.4.1. Classification of elements according to their optimal number of integration points

The optimization procedure of the number of integration points is given as follows:

(1) Data preparation phase: the element stiffness matrix is numerically calculated using the Gauss–Legendre quadrature rule, respectively, with one to $q_{\max}$ integration points per coordinate axis and the corresponding error is evaluated by Eq. (17). A set of the minimum number of integration points $\left\{n_{opt}, m_{opt}, l_{opt}\right\}$, which give smaller error than the value set in advance, is obtained. Thus, many data pairs $(\{e-parameters\}, \left\{n_{opt}, m_{opt}, l_{opt}\right\})$ are collected.

(2) Training phase: Using the data prepared in the data preparation phase above, a classifier is constructed by deep learning, which consists of the element parameters $\{e-parameters\}$ as input and the set of optimal number of integration points $\left\{n_{opt}, m_{opt}, l_{opt}\right\}$ as output or the teacher signal.

(3) Application phase: The trained classifier constructed in the training phase is implemented on the numerical quadrature process of the finite element analysis code. The numerical integration of each element stiffness matrix is respectively performed using the Gauss–Legendre quadrature with the corresponding set of the optimal number of integration points indicated by the implemented classifier.

Here, the element parameters that identify the element are written as $\{e-parameters\}$.

3.4.2. Determination of optimal weights for numerical quadrature in element stiffness matrix calculation

Here again, the element parameters that identify the element, such as coordinates of the nodes in the element, are written as $\{e-parameters\}$, and the prescribed number of integration points as $\left\{q_\xi, q_\eta, q_\zeta\right\}$, each component of which corresponds to that for a coordinate axis.

(1) Data preparation phase: With the set of numbers of integration points being $\{q_{\max}, q_{\max}, q_{\max}\}$ for each of elements, the element stiffness matrix is calculated using the Gauss–Legendre quadrature with the standard weights $\left\{H_{i,j,k}\right\}$, and the result is regarded as the true or correct value of the integral. With the prescribed set of numbers of the integration points $\left\{q_\xi, q_\eta, q_\zeta\right\}$, the integral of the same matrix is repeatedly performed using the Gauss–Legendre quadrature, not with the standard $\left\{H_{i,j,k}\right\}$ but with different $\left\{w_{i,j,k} \times H_{i,j,k}\right\}$. For each result, the error defined in Eq. (17) is evaluated. Thus, $\left\{w_{i,j,k}^{opt}\right\}$ is determined, and the data pairs ($\{e-parameters\}$, $\left\{w_{i,j,k}^{opt}\right\}$) are collected.

(2) Training phase: Using the data prepared in the data preparation phase above, a feedforward neural network is constructed with deep learning, assuming the element parameters $\{e-parameters\}$ as input and the set of optimal correction factors $\left\{w_{i,j,k}^{opt}\right\}$ as output or the teacher signal.

(3) Application phase: The trained feed forward neural network produced in the training phase is implemented on the numerical quadrature process of the finite element analysis code. When the $\{e-parameters\}$ of an element is supplied to the neural network, it outputs the optimal correction factors $\left\{w_{i,j,k}^{opt}\right\}$ of the element, and the integral of the element stiffness matrix of the element is achieved using the Gauss–Legendre quadrature based on Eq. (20) with the optimal correction factors $\left\{w_{i,j,k}^{opt}\right\}$.

3.5. DL-enhanced numerical quadrature: Numerical example

3.5.1. Classification of elements according to their optimal number of integration points

Here, we show an application of the proposed method to the integral of the element stiffness matrix of the eight-node linear hexahedral element, where the convergence properties of the quadrature are evaluated among elements with different set of node locations.

3.5.1.1. Data preparation phase. As shown in Fig. 6, the node A is located at the origin of the 3D coordinates, the node B on the x axis, and the node D on the x-y plane. An arbitrary eight-node hexahedral linear element can be converted to this standardized arrangement by the following step-by-step operations (see Fig. 7):

(1) The element is located as the node A being at the origin (Fig. 7(a)).

(2) It is rotated about the z axis to locate the node B on the x-z plane (Fig. 7(b)).

(3) It is further rotated about the y axis to locate the node B on the x axis (Fig. 7(c)).

(4) It is then rotated about the x axis to locate the node D on the x-y plane (Fig. 7(d)).

(5) Finally, the element is transformed similarly by $\frac{1}{l_0}$ times, where l_0 is the average length of the edge AB and the edge AD of the element.

Since the convergence properties of the quadrature of the element integration depend only on the shape of the element and are independent on its size, location and orientation, only the shape of the element is considered as the key parameter, while other redundant information is eliminated. Here, a number of elements of various shapes are generated with the shape of a unit cube being the reference arrangement of node, where the coordinates of nodes are changed using a uniform random number r of the range [0, 1] and the maximum amount of change d as follows;

$$A\,(0,0,0)\,,\quad B\,(1\pm rd,0,0)\,,\quad C\,(1\pm rd,1\pm rd,\pm rd)\,,\quad D\,(\pm rd,1\pm rd,0)\,,$$

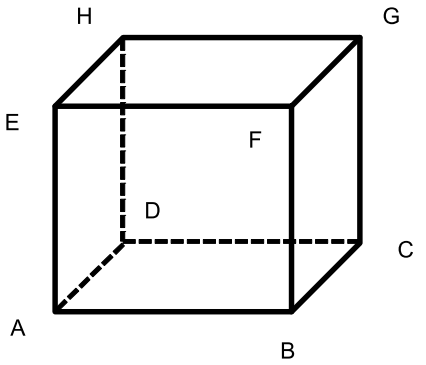


Fig. 6. 8-noded linear solid element.

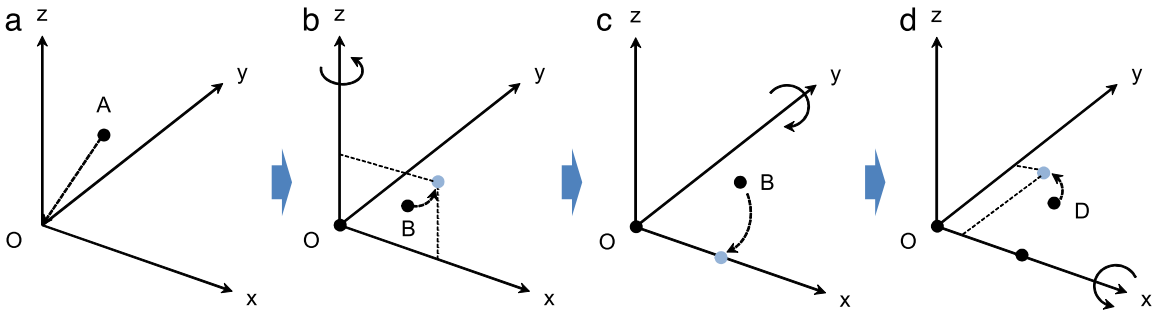


Fig. 7. Nodes A, B and D of any 8-noded element are shifted to x-y plane by translation (a) and rotations (b), (c) and (d).

$E\,(\pm rd, \pm rd, 1 \pm rd)$, $F\,(1 \pm rd, \pm rd, 1 \pm rd)$, $G\,(1 \pm rd, 1 \pm rd, 1 \pm rd)$, and
$H\,(\pm rd, 1 \pm rd, 1 \pm rd)$.

Here, the maximum amount of change in the coordinate values is selected from $d = 0.1, 0.2, 0.3, 0.4$ and 0.5. It is noted that, if the coordinates of each node in an element are independently changed, elements of heavily distorted or inappropriate concave shape may be generated. In order to exclude these shapes, we impose the following restrictions on the process:

(1) The length of each edge is in the range of 0.5 to 2.0.

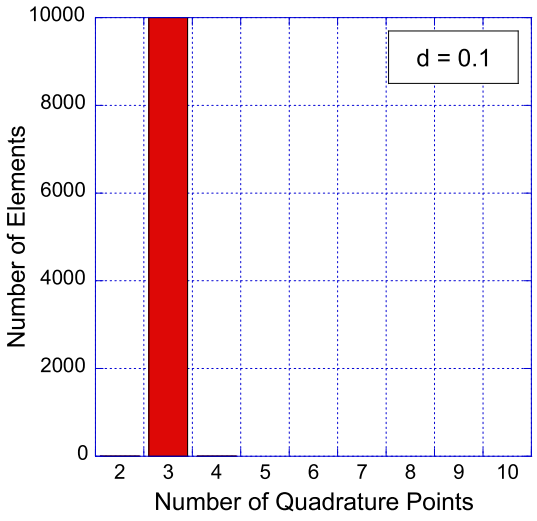
(2) The angle between adjacent faces in the element is in the range of 60° to 120°.

Convergence properties of each of the elements generated above are evaluated according to Eq. (17) and the results are shown in Fig. 8. Fig. 8(a), (b), and (c) show the distribution of the minimum number of the integration points, required for the error to be smaller than 1.0×10^{-3} in the cases of $d = 0.1$, $d = 0.3$ and $d = 0.5$, respectively. It can be seen from the figures that, as the shape is distorted, more integration points are needed.

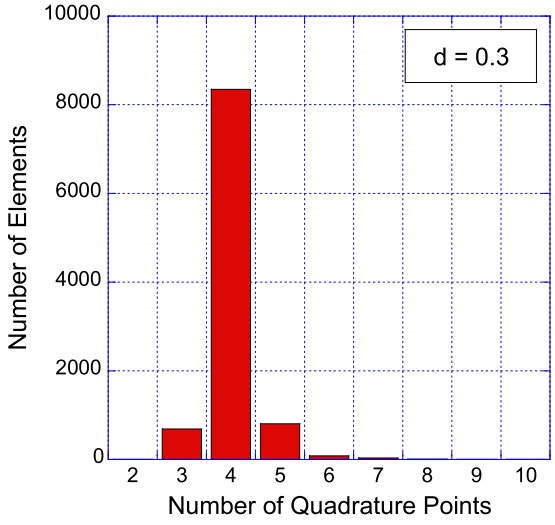
3.5.1.2. Training phase. A total of 10,000 data pairs, each of which consists of coordinates of nodes in an element and the minimum number of integration points required to integrate it with the error below 1.0×10^{-3}, are generated from a total of 10,000 elements consisting of five groups, each of which has 2000 elements generated setting d as 0.1, 0.2, 0.3, 0.4 and 0.5, respectively. Using these data pairs, a neural network is trained to classify elements according to the minimum number of integration points required. Here, the input and the teacher signals for the neural network are assumed as follows:

Input data: Coordinate values of the nodes in an element. In this case, it results in 18 coordinate values out of 24 coordinate values of eight nodes excluding following 6 values: x, y and z values of node A, y and z values of node B, z value of node D in Fig. 6. Thus the network to be constructed has 18 units in the input layer of the neural network.

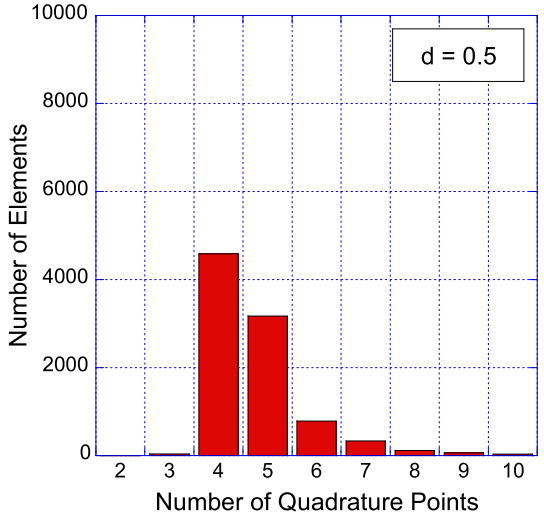
Teacher signal: The minimum number of integration points that makes the error given in Eq. (17) smaller than 1.0×10^{-3}.



(a) $d = 0.1$.



(b) $d = 0.3$.



(c) $d = 0.5$.

Fig. 8. Number of quadrature points required to converge within prescribed error (10^{-3}) versus distribution of numbers of elements.

Out of 10,000 data pairs, we employ 5000 pairs for training patterns, whereas the rest for test patterns to check the generalization capability of the trained neural network. To find the appropriate network architecture, several neural networks with different number of hidden layers and units in the hidden layers are trained using the error back propagation method. The training results are shown in Table 1. It can be seen from the table that the classification accuracy over 80 % is obtained with three hidden layers and 50 units per hidden layer or beyond.

3.5.1.3. Application phase. Table 2 shows the results of the classification for both training and test patterns, obtained by the best trained neural network with three hidden layers and 50 units per hidden layer. It can be seen from the table that, even when the trained neural network fails to answer the correct number of integration points, it usually gives an answer close to the correct one.

We conclude that deep learning makes it possible to estimate the minimum number of integration points to obtain a needed accuracy on any given element, which may result in the element stiffness matrix with high accuracy.

Table 1
Classification accuracy versus network architecture.

Number of hidden layers	Number of units per hidden layer	Results (%)	
		Training patterns	Test patterns
1	50	91.4	79.2
2	50	95.9	80.4
3	50	98.6	81.6
4	50	97.8	81.4
5	50	97.4	81.2

Table 2
Results of classification.

(a) Training patterns

		Number of quadrature points (estimated by neural network)								
		2	3	4	5	6	7	8	9	Total
Number of quadrature points (correct)	2		5							5
	3		1553	9						1562
	4		7	2548	2					2557
	5			20	616					636
	6			5	2	153	2			162
	7			2	5	1	46	1		55
	8				5	1		15		21
	9				1	1				2

(b) Test patterns

		Number of quadrature points (estimated by neural network)								
		2	3	4	5	6	7	8	9	Total
Number of quadrature points (correct)	2		5							5
	3		1430	123						1553
	4		135	2222	201	15	1			2574
	5			206	386	56	7	1		656
	6			16	70	36	13			135
	7			1	29	19	6	1		56
	8				10	8	3			21
	9				3	2				5

3.5.2. Determination of optimal weights for numerical quadrature in element stiffness matrix calculation

3.5.2.1. Data preparation phase. Elements are set as in Section 3.5.1, and the quadrature of the element integral is performed with 8 integration points. For each element, a number of correction factors $\{w_{i,j,k}\}$ are tested and the best one $\left\{w^{opt}_{i,j,k}\right\}$ for the element is determined based on the error defined in Eq. (17), and then the error reduction ratio R_{Error} is defined as follows,

$$R_{Error} = \frac{Error \ \text{obtained using} \ \left\{w^{opt}_{i,j,k}\right\}}{Error \ \text{obtained using default quadrature weights}} \tag{21}$$

where $Error$ on the right hand side is the same as that defined in Eq. (17).

A total of 20,000 elements consisting of five element groups, each of which has 4000 elements generated by assuming $d = 0.1,\ 0.2,\ 0.3,\ 0.4,\ and 0.5$, respectively, are tested. For each element, the quadrature of the element stiffness matrix is performed using one million sets of correction factors $\{w_{i,j,k}\}$ generated by random numbers in the range of $[0.95,\ 1.05]$ for each of eight integration points, and the error reduction ratio with $\left\{w^{opt}_{i,j,k}\right\}$ is obtained. Table 3 shows the achieved results, where Category A means that an element cannot find a set of correction factors for improving accuracy of the quadrature among one million sets of correction factors, otherwise Category B. The table also shows that for elements with little distortion in shape it is difficult to reduce the error by the correction factors.

Table 3
Number of elements that belong to category A ($R_{Error} \geq 1.0$) or B ($R_{Error} < 1.0$) for five element groups, respectively.

		Category		
		A	B	Total
d	0.1	4000	0	4000
	0.2	3973	27	4000
	0.3	3350	650	4000
	0.4	2320	1680	4000
	0.5	1489	2511	4000

Table 4
Results of classification into two categories A and B based on whether or not accuracy of quadrature is improved using some correction factors.

(a) Training patterns

		Category (estimated by neural network)		
		A	B	Total
Category (correct)	A	3707	29	3736
	B	70	1194	1264

(b) Test patterns

		Category (estimated by neural network)		
		A	B	Total
Category (correct)	A	3682	155	3837
	B	224	939	1163

To solve this problem, we adopt the two-stage optimization as follows. In the first stage, by using a classification neural network, each element is classified into Category A or B according to whether it can reduce the error with some correction factors or not, and then, in the second stage, by using a regression neural network, the best correction factors $\left\{w_{i,j,k}^{opt}\right\}$ are identified only for the elements categorized as Category B in the above.

3.5.2.2. Training phase. As the first stage, a neural network to judge whether error reduction is possible or not in the element given is tuned by using a total of 10,000 patterns selected from the 20,000 patterns described above. The input and the teacher signal of the neural network are as follows:

Input data: Coordinate values of the nodes in an element, where teacher signal: 1 for $R_{Error} < 1$, and 0 for $R_{Error} \geq 1$. Note that the neural network has one unit in the output layer.

Out of the 10,000 patterns, 5000 patterns are selected for training, and the others for test patterns to estimate generalization capability. The maximum number of epochs is set to 30,000. Many cycles of trainings are performed with various learning parameters, showing that the best result is obtained from the neural network with four hidden layers and 30 units per hidden layer as shown in Table 4. It is noted that the classification accuracy, i.e. the rate of correct answer, obtained is as high as 98% and 92% for the training and the test patterns, respectively. Then, as the second stage, a neural network to identify a set of optimized correction factors for a given element is trained by using a total of 10,000 patterns obtained from the elements with $R_{Error} < 1$. The input and the teacher signal of the neural network are as follows:

Input data: Coordinate values of the nodes in an element.

Teacher signal: Eight correction factors, each corresponding to one of eight integration points. Note that the neural network has eight units in the output layer.

The 5000 patterns out of the 10,000 patterns are selected again for training, and the others for test patterns to estimate generalization capability. Setting the maximum number of epochs being 30,000, the best result is obtained from the neural network with five hidden layers and 50 units per hidden layer. Fig. 9 shows the distribution of error in identifying the value of each correction factor obtained by the network.

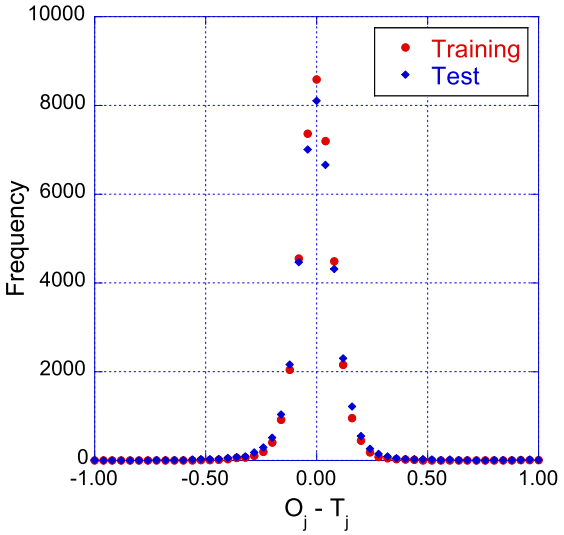
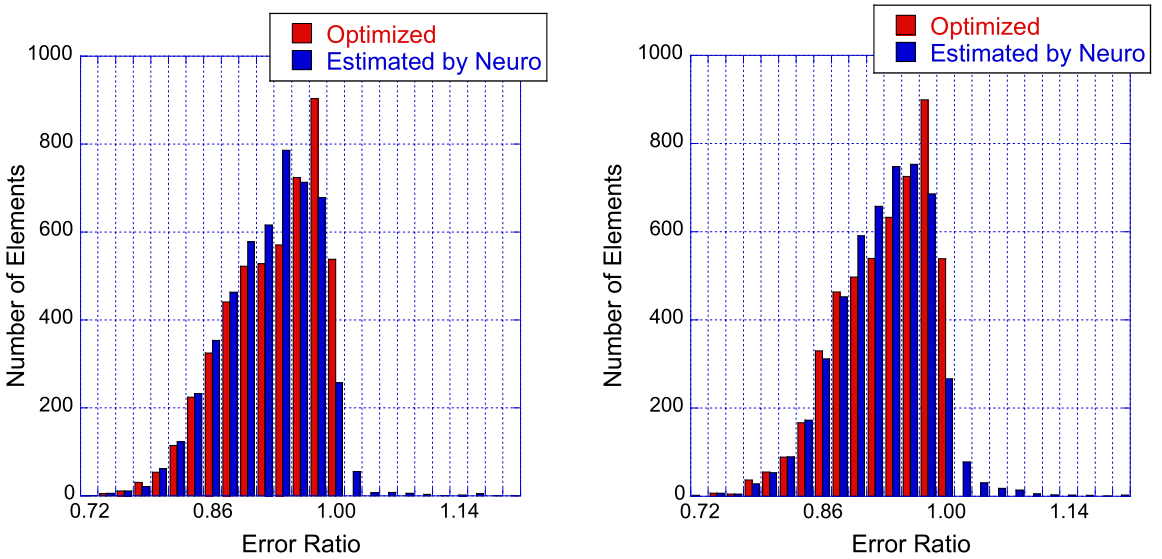


Fig. 9. Distributions of estimation errors for [0, 1] normalized correction factors.



(a) Training patterns.

(b) Test patterns.

Fig. 10. Distributions of error ratios defined by Eq. (21), when using correction factors estimated by deep learning.

3.5.2.3. Application phase. Fig. 10 shows the distribution of the error reduction ratio obtained in the quadrature of the element stiffness matrix using correction factors estimated by the trained neural network, where Figs. 10 (a) and (b) show the distribution obtained from training and test patterns, respectively. The distribution of error reduction ratio obtained using the corresponding optimized correction factors, i.e. the teacher signal, is also shown in the figures as a reference for comparison. It is clearly shown in the above figures that even with the correction factors estimated by the trained neural network, the improvement of the accuracy in the quadrature can be achieved to almost the same extent as with the best, optimized correction factors. In other words, the accuracy in the quadrature is improved in 99% of the training patterns and 97% of the test patterns.

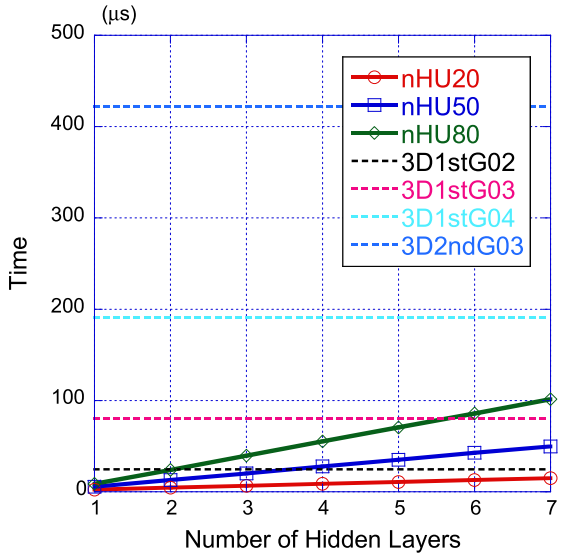


Fig. 11. Computational costs in the application phase. Three solid lines show the time required for the estimation by the trained network with eighteen input units, eight output units, and twenty (red), fifty (blue) and eighty (green) units per hidden layer. Four dotted lines show the computing time required to calculate one element stiffness matrix for four cases: three types of eight-node linear hexahedral elements with $2 \times 2 \times 2$, $3 \times 3 \times 3$ and $4 \times 4 \times 4$ quadrature points (designated respectively as 3D1stG02, 3D1stG03 and 3D1stG04) and a twenty-node quadratic hexahedral element with $3 \times 3 \times 3$ quadrature points (3D2ndG03). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

It is concluded that, by using deep learning, it will be very promising to improve the accuracy of the numerical quadrature of the element stiffness matrix. In other words, given the number of integration points and their coordinates, we can obtain more accurate quadrature with correction factors estimated by the trained neural network than with the standard Gauss–Legendre formula.

As for the computational cost of the estimation by the trained neural network in the application phase, Fig. 11 shows the time required to process single input pattern to obtain corresponding output data for different numbers of hidden layers. The number of input units and output units are set to eighteen and eight, respectively, which are identical to those of the network used in this application. The number of hidden units per layer is set to 20, 50 (identical to that used in this application) and 80. The computing time required to calculate one element stiffness matrix by the standard Gauss–Legendre quadrature is also shown by the dotted lines for comparison for four cases: three types of eight-node linear hexahedral elements with $2 \times 2 \times 2$ quadrature points (designated as 3D1stG02 in Fig. 11), $3 \times 3 \times 3$ quadrature points and $4 \times 4 \times 4$ quadrature points, and a twenty-node quadratic hexahedral element with $3 \times 3 \times 3$ quadrature points (3D2ndG03). All the timings are measured on the intel Xeon E5507 processor. It is shown in Fig. 11 that the estimation by the neural network of the five hidden layers and 50 units per layer, which is the same as the one adopted here, takes almost as long computation time as the numerical quadrature of single eight-node linear hexahedral element with $2 \times 2 \times 2$ quadrature points. The reduction of estimation time in the application phase is an important issue in our method, which is discussed in the following chapter.

4. Discussion

It is well recognized that deep learning may expand the scope of application of the feed forward neural networks to computational mechanics. On the other hand, as the number of hidden layers and the number of units per layer of the neural network significantly increase, a large amount of computational power is required for its training. In the training, however, there is little restriction on the available computational resources and computing time because the training is done independently from the application itself that requires a fast and often real-time computing, suggesting that one can make use of deep learning specific computers, such as GPU clusters, to manage the growth of the computational load in the training phase.

A large scale neural network tuned by deep learning is, as a matter of course, employed in the application phase, where available computers are usually much more limited than in the training phase. Thus, the growth in the scale of the neural networks is regarded as more serious and severe issue in the application phase than in the training phase.

Here, we address this issue, a faster computation for the application phase in the DL-enhanced computational mechanics from the viewpoint of making use of low-precision computation as follows. As is well known, to solve partial differential equations numerically has always been the main part of computational mechanics, which is usually performed with double precision (FP64) computation. Then, computing speed with the double precision calculation is possibly the most important factor to select computers for computational mechanics applications. In deep learning, on the other hand, it has been recognized that even the lower-precision computation performs well particularly in classification problems. Several studies have been published on this issue: a test of deep learning with limited numerical precision [112], a comparative evaluation of low precision storage for deep learning [113], a test of training with connection weights and activations constrained to 1 or -1 [114]. It is also known that adopting low precision format for numerical data directly reduces the amount of memory space required to store the data, which will make the calculation faster and reduce the computational time together with the arithmetic unit specific to the low precision format. When using some GPUs equipped with the arithmetic units specific to calculation of number of half precision format (FP16), we can expect almost doubly faster calculations for deep learning applications with the FP16 format than with FP32. Besides GPUs, FPGA (Field Programmable Gate Array) [115], in which low precision arithmetic circuits dedicated to deep learning calculations are implemented, is expected to perform well. Using the fixed point real number format rather than the floating point one, arithmetic circuits specific to the low-precision format, implemented in the FPGA, would become much more compact and perform much faster than the corresponding circuits for floating point numbers.

The application of deep learning to computational mechanics will be more important in the regression-type problems than those of the classification-type. Therefore, we should check how far the numerical precision is needed in each problem.

Here, we focus on the application phase because the requirement on the calculation speed is more serious in the application phase than in the training phase. As for the latter phase, ordinary arithmetic operations with real numbers of single precision floating point format are employed partly because the training requires gradient calculations that are usually sensitive to precision and partly because it is independently performed with the application phase. However, as arithmetic operations with double precision numbers are still used in the core process of the finite element analysis, numerical data of double precision are often converted to those of low precision, where an efficient conversion method should be prepared.

In order to study the usefulness of the low precision arithmetic in the proposed DL-enhanced computational mechanics, we have tested the influence of the low precision arithmetic on the accuracy of the estimation of the trained neural network for the classification and that for the regression discussed in Section 3.

Fig. 12 shows a method of precision reduction from a single precision floating point number to that of floating point with an arbitrary precision, where the most popular IEEE754 single precision floating point format (FP32) is used, which consists of single sign bit, eight exponent bits and twenty three fraction bits. Shifting the original data n bits to the right and then n bits to the left, the lower n bits of the fraction part are filled with 0, reducing the number of significant digits. The above operation, named as a PLP method (Pseudo Low Precision method), is to be carried out after every arithmetic operation that is executed using the arithmetic unit designed for FP32 or FP64 data. Since the PLP, requiring just the bit shift operation, is easy to implement, we can perform simple evaluation of the influence of the precision reduction on the results estimated by the neural networks.

Here, we have studied with the PLP how further the numerical precision of the data can be reduced without significant degradation of accuracy of the classification or the regression estimated by the neural network trained by deep learning. Fig. 13 shows the accuracy of classification obtained with varying the number of shifted bits for the classification neural network of three hidden layers and 50 units per hidden layer trained by deep learning in Section 3.5.1, where the reduction of the precision by the PLP is applied to all the connection weights, biases and input data of the neural network at every arithmetic operation. The figure indicates that a significant reduction of effective digits has little influence on the accuracy of the classification. In other words, the good accuracy of classification as high as more than 80%, only a little lower than 81.6% for the original data of full precision, is obtained when using the PLP with 16 bits. Table 5 shows the classification results obtained by the trained neural network using the PLP with 16-bit shift, which are almost equivalent to those obtained with original full numerical precision shown in Table 2(b).

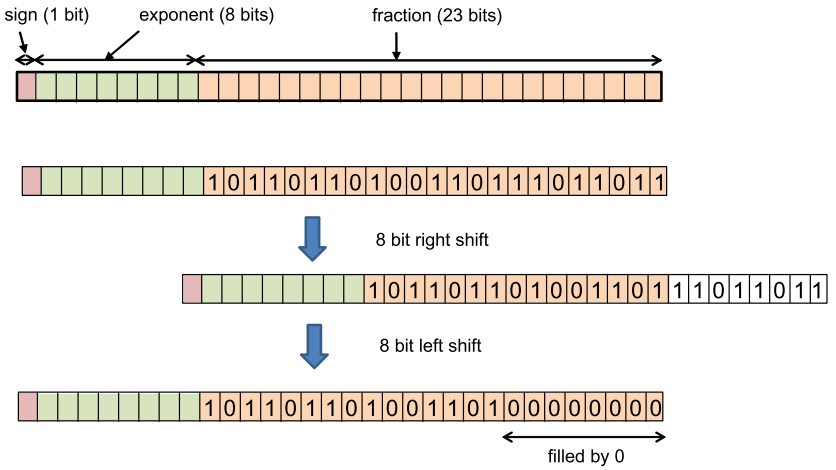


Fig. 12. Pseudo low precision (PLP) operation for a floating point number with IEEE 754 single precision (32 bit) floating point format, performing n-bit shift operations twice, first to the right and then to the left, to fill 0 s in the least significant n bits.

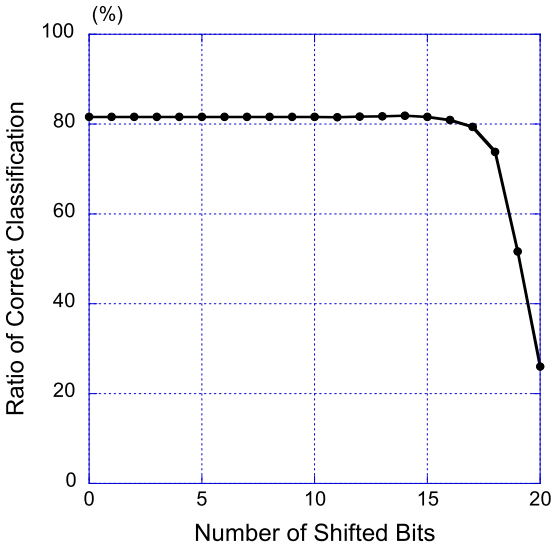


Fig. 13. Classification accuracy: ratio of correct classification versus number of bits shifted in PLP operations. It is seen that with small amount of bits shifted, no deterioration occurs in classification accuracy.

Besides the classification problems above, we have tested the usefulness of the low precision arithmetic in the regression problems, where teacher signals are real-valued. Fig. 14 shows the accuracy of the estimation measured by sum of squared error by using the PLP, varying the number of shifted bits for the neural network of five hidden layers and 50 units per hidden layer for identifying $\left\{w^{opt}_{i,j,k}\right\}$ trained by deep learning given in Section 3.5.2. It can be seen from the figure that the sum of squared error defined in Eq. (3) does not grow for the PLP with less than 15 bits but does rapidly with more than 16 bits, implying that the numerical precision of only 8 bit in fraction suffices to the neural network tested.

In the PLP used in this research, the exponent part remains unchanged, thus the dynamic range among the original data also does. If the dynamic range among the original data including connection weights, biases and input data is small, it becomes possible to adopt the fixed point real number instead of the floating point one, which results in the possibility to develop fast arithmetic circuits based on the fixed point number format on a FPGA, specialized for the trained feedforward neural network. As for the reduction of the dynamic range among the connection weights and

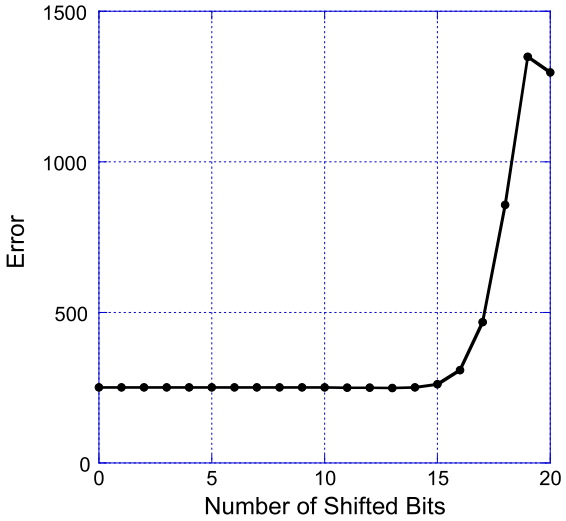


Fig. 14. Identification error: sum of squared error for test patterns versus number of bits shifted in PLP operations. It is seen that with small amount of bits shifted, identification error does not grow.

Table 5
Results of classification for test patterns when tested with 16 bit PLP.

		Number of quadrature points (estimated by neural network)								
		2	3	4	5	6	7	8	9	Total
Number of quadrature points (correct)	2									0
	3		1397	156						1553
	4		124	2275	170	5				2574
	5			271	345	33	6	1		656
	6			22	77	24	12			135
	7			5	34	13	4			56
	8			1	11	6	3			21
	9				3	2				5

biases, some regularization methods for training of neural networks are considered to be worth trying: the weight decay training method [8] and the training method with the connection weight kept below the prescribed value [116].

As discussed above, low precision arithmetic suffices for the computation related to the trained neural network in the proposed DL-enhanced computational mechanics, and, together with the arithmetic circuits specifically developed for low-precision numerical data, it is expected to achieve significant performance. For a future work, it is important to develop a hardware that can perform well not only for double precision computation but also for low precision computation, the former corresponding to the core part of the finite element method, and the latter to the deep learning part.

5. Conclusion

This paper focuses on the rule extraction capability of deep learning, showing the possibility of its new application to the computational mechanics in which conventional neural networks have never been successful, including the optimization of numerical quadrature of the FEM stiffness matrix.

The proposed method, employing the strong classification and regression capability of deep learning, optimizes the numerical quadrature rule for the FEM element stiffness matrix in the element-by-element basis. It can perform numerical quadrature more accurately than the standard Gauss–Legendre rule with the same number of integration points. It also can perform numerical quadrature with minimum number of integration points.

The proposed numerical quadrature rule enhanced by deep learning is applicable to various fields of computational mechanics. It is totally different from most of other methods that heavily depend on the mathematical structure specific to the target integrand with limited applicability.

A general framework for the application of deep learning to the computational mechanics is also proposed. Deep learning, which is based on the hierarchical neural network with a large number of hidden layers and hidden units has overstepped the limits of the conventional neural network, widening its application scope to a large extent.

References

[1] G. Yagawa, A. Yoshioka, S. Yoshimura, N. Soneda, A parallel finite element method with a supercomputer network, Comput. Struct. 47 (3) (1993) 407–418.

[2] K. Garatani, K. Nakajima, H. Okuda, G. Yagawa, Three-dimensional elasto-static analysis of 100 million degrees of freedom, Adv. Eng. Softw. 32 (2001) 511–518.

[3] H. Akiba, T. Ohyama, Y. Shibata, K. Yuyama, Y. Katai, R. Takeuchi, T. Hoshino, S. Yoshimura, H. Noguchi, M. Gupta, J.A. Gunnels, V. Austel, Y. Sabharwal, R. Garg, S. Kato, T. Kawakami, S. Todokoro, J. Ikeda, Large scale drop impact analysis of mobile phone using ADVC on Blue Gene/L, in: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, November, 2006, Tampa, Florida http://dx.doi.org/10.1145/1188455.1188503.

[4] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[5] K.P. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012.

[6] Q.V. Le, M.A. Ranzato, R. Monga, M. Devin, K. Chen, G.S. Corrado, J. Dean, A.Y. Ng, Building high-level features using large scale unsupervised learning, in: Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, May 2013, pp. 8595–8598.

[7] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of Go with deep neural networks and tree search, Nature 529 (2016) 484–489.

[8] S. Heykin, Neural Networks: A Comprehensive Foundation, Prentice Hall, 1999.

[9] D.E. Goldberg, Genetic algorithms in search, in: Optimization & Machine Learning, Addison-Wesley, 1989.

[10] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs (Third, Revised and Extended Edition), Springer, 1996.

[11] J.R. Koza, Genetic Programming: On the Programming of Computers By Means of Natural Selection, MIT Press, 1992.

[12] J.R. Koza, Genetic Programming II: Automatic Discovery of Reusable Programs, MIT Press, 1994.

[13] A. Amirjanov, Investigation of a changing range genetic algorithm in noisy environments, Internat. J. Numer. Methods Engrg. 73 (2008) 26–46.

[14] D. Rabinovich, D. Givoli, S. Vigdergauz, XFEM-based crack detection scheme using a genetic algorithm, Internat. J. Numer. Methods Engrg. 71 (2007) 1051–1080.

[15] R.E. Smith, B.A. Dike, R.K. Mehra, B. Ravichandran, A. El-Fallah, Classifier systems in combat: two-sided learning of maneuvers for advanced fighter aircraft, Comput. Methods Appl. Mech. Engrg. 186 (2000) 421–437.

[16] A. Rovira, M. Valdes, J. Casanova, A new methodology to solve non-linear equation systems using genetic algorithms. Application to combined cycle gas turbine simulation, Internat. J. Numer. Methods Engrg. 63 (2005) 1424–1435.

[17] T. Furukawa, G. Yagawa, Inelastic constitutive parameter identification using an evolutionary algorithm with continuous individuals, Internat. J. Numer. Methods Engrg. 40 (1997) 1071–1090.

[18] M.B. Shim, M.W. Suh, T. Furukawa, G. Yagawa, S. Yoshimura, Pareto-based continuous evolutionary algorithms for multiobjective optimization, Eng. Comput. 19 (1) (2002) 22–48.

[19] D. Ishihara, M.J. Jeong, S. Yoshimura, G. Yagawa, Design window search using continuous evolutionary algorithm and clustering - its application to shape design of microelectrostatic actuator, Comput. Struct. 80 (2002) 2469–2481.

[20] M. Wang, D. Dutta, K. Kim, J.C. Brigham, A computationally efficient approach for inverse material characterization combining Gappy POD with direct inversion, Comput. Methods Appl. Mech. Eng. 286 (2015) 373–393.

[21] B. Peherstorfer, K. Willcox, Dynamic data-driven reduced-order models, Comput. Methods Appl. Mech. Eng. 291 (2015) 21–41.

[22] D. Wirtz, N. Karajan, B. Haasdonk, Surrogate modeling of multiscale models using kernel methods, Internat. J. Numer. Methods Engrg. 101 (2014) 1–28.

[23] B. Peherstorfer, T. Cui, Y. Marzouk, K. Willcox, Multifidelity importance sampling, Comput. Methods Appl. Mech. Eng. 300 (2016) 490–509.

[24] R.S. Parpinelli, F.R. Teodoro, H.S. Lopes, A comparison of swarm intelligence algorithms for structural engineering optimization, Internat. J. Numer. Methods Engrg. 91 (2012) 666–684.

[25] I.N. Vieira, B.S.L. Pires de Lima, B.P. Jacob, Bio-inspired algorithms for the optimization of offshore oil production systems, Internat. J. Numer. Methods Engrg. 91 (2012) 1023–1044.

[26] M. Zimmermann, J.E. von Hoessle, Computing solution spaces for robust design, Internat. J. Numer. Methods Engrg. 94 (2013) 290–307.

[27] D. Kohler, Y.M. Marzouk, J. Muller, U. Wever, A new network approach to Bayesian inference in partial differential equations, Internat. J. Numer. Methods Engrg. 104 (2015) 313–329.

[28] I.M. Franck, P.S. Koutsourelakis, Sparse variational Bayesian approximations for nonlinear inverse problems: Applications in nonlinear elastography, Comput. Methods Appl. Mech. Eng. 299 (2016) 215–244.

[29] L. Tan, S.R. Awade, Response classification of simple polycrystalline microstructures, Comput. Methods Appl. Mech. Eng. 197 (2008) 1397–1409.
[30] P.M. Congedo, C. Corre, J.-M. Martinez, Shape optimization of an airfoil in a BZT flow with multiple-source uncertainties, Comput. Methods Appl. Mech. Eng. 200 (2011) 216–232.
[31] S. Sankaran, L. Grady, C.A. Taylor, Impact of geometric uncertainty on hemodynamic simulations using machine learning, Comput. Methods Appl. Mech. Eng. 297 (2015) 167–190.
[32] T. Kirchdoerfer, M. Ortiz, Data-driven computational mechanics, Comput. Methods Appl. Mech. Eng. 304 (2016) 81–101.
[33] G. Yagawa, H. Okuda, Neural networks in computational mechanics, Arch. Comput. Methods Eng. 3 (4) (1996) 435–512.
[34] G. Yagawa, A. Matsuda, H. Kawate, Neural network approach to estimate stable crack growth in welded specimens, Int. J. Press. Vessels Pip. 63 (1995) 303–313.
[35] S. Yoshimura, Y. Saito, G. Yagawa, Identification of two dissimilar surface cracks hidden in solid using neural networks and computational mechanics, Comput. Model. Simul. Eng. 1 (1996) 477–491.
[36] A. Oishi, S. Yoshimura, A new local contact search method using a multi-layer neural network, Comput. Model. Eng. Sci. 21 (2) (2007) 93–103.
[37] J.H. Kim, Y.H. Kim, A predictor–corrector method for structural nonlinear analysis, Comput. Methods Appl. Mech. Eng. 191 (2001) 959–974.
[38] R. Lopez, E. Balsa-Canto, E. Onate, Neural networks for variational problems in engineering, Internat. J. Numer. Methods Engrg. 75 (2008) 1341–1360.
[39] S. Yoshimura, A. Matsuda, G. Yagawa, New regularization by transformation for neural network based inverse analyses and its application to structure identification, Internat. J. Numer. Methods Engrg. 39 (1996) 3953–3968.
[40] T. Furukawa, G. Yagawa, Implicit constitutive modelling for viscoplasticity using neural networks, Internat. J. Numer. Methods Engrg. 43 (1998) 195–219.
[41] N. Huber, Ch. Tsakmakis, A neural network tool for identifying the material parameters of a finite deformation viscoplasticity model with static recovery, Comput. Methods Appl. Mech. Eng. 191 (2001) 353–384.
[42] M. Lefik, B.A. Schrefler, Artificial neural network as an incremental non-linear constitutive model for a finite element code, Comput. Methods Appl. Mech. Eng. 192 (2003) 3265–3283.
[43] S. Jung, J. Ghaboussi, Characterizing rate-dependent material behaviors in self-learning simulation, Comput. Methods Appl. Mech. Eng. 196 (2006) 608–619.
[44] H. Man, T. Furukawa, Neural network constitutive modelling for non-linear characterization of anisotropic materials, Internat. J. Numer. Methods Engrg. 85 (2011) 939–957.
[45] M. Lefik, D.P. Boso, B.A. Schrefler, Artificial neural networks in numerical modelling of composites, Comput. Methods Appl. Mech. Eng. 198 (2009) 1785–1804.
[46] J. Ghaboussi, D.A. Pecknold, M. Zhang, R. Haj-Ali, Autoprogressive training of neural network constitutive models, Internat. J. Numer. Methods Engrg. 42 (1998) 105–126.
[47] M.S. Al-Haik, H. Garmestani, I.M. Navon, Truncated-Newton training algorithm for neurocomputational viscoplastic model, Comput. Methods Appl. Mech. Eng. 192 (2003) 2249–2267.
[48] Y.M.A. Hashash, S. Jung, J. Ghaboussi, Numerical implementation of a network based material model in finite element analysis, Internat. J. Numer. Methods Engrg. 59 (2004) 989–1005.
[49] M. Oeser, S. Freitag, Modeling of materials with fading memory using neural networks, Internat. J. Numer. Methods Engrg. 78 (2009) 843–862.
[50] Y. Ootao, R. Kawamura, Y. Tanigawa, Optimization of material composition of nonhomogeneous hollow sphere for thermal stress relaxation making use of neural network, Comput. Methods Appl. Mech. Eng. 180 (1999) 185–201.
[51] D. Gawin, M. Lefik, B.A. Schrefler, ANN approach to sorption hysteresis within a coupled hygro-thermo-mechanical FE analysis, Internat. J. Numer. Methods Engrg. 50 (2001) 299–323.
[52] G.J. Yun, J. Ghaboussi, A.S. Elnashai, Self-learning simulation method for inverse nonlinear modeling of cyclic behavior of connections, Comput. Methods Appl. Mech. Eng. 197 (2008) 2836–2857.
[53] G.E. Stavroulakis, H. Antes, Neural crack identification in steady state elastodynamics, Comput. Methods Appl. Mech. Eng. 165 (1998) 129–146.
[54] S.W. Liu, J.H. Huang, J.C. Sung, C.C. Lee, Detection of cracks using neural networks and computational mechanics, Comput. Methods Appl. Mech. Eng. 191 (2002) 2831–2845.
[55] A. Oishi, K. Yamada, S. Yoshimura, G. Yagawa, Quantitative nondestructive evaluation with ultrasonic method using neural networks and computational mechanics, Comput. Mech. 15 (6) (1995) 521–533.
[56] A. Oishi, K. Yamada, S. Yoshimura, G. Yagawa, S. Nagai, Y. Matsuda, Neural network-based inverse analysis for defect identification with laser ultrasonics, Res. Nondestruct. Eval. 13 (2001) 79–95.
[57] N.S. Mera, L. Elliott, D.B. Ingham, The use of neural network approximation models to speed up the optimization process in electrical impedance tomography, Comput. Methods Appl. Mech. Eng. 197 (2007) 103–114.
[58] J. Zacharias, C. Hartmann, A. Delgado, Damage detection on crates of beverages by artificial neural networks trained with finite-element data, Comput. Methods Appl. Mech. Eng. 193 (2004) 561–574.
[59] N. Garijo, J. Martinez, J.M. Garcia-Aznar, M.A. Perez, Computational evaluation of different numerical tools for the prediction of proximal femur loads from bone morphology, Comput. Methods Appl. Mech. Eng. 268 (2014) 437–450.
[60] M. Papadrakakis, N.D. Lagaros, Y. Tsompanakis, Structural optimization using evolution strategies and neural networks, Comput. Methods Appl. Mech. Eng. 156 (1998) 309–333.

[61] C. Polini, A. Giurgevich, L. Onesti, V. Pediroda, Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for a complex design problem in fluid dynamics, Comput. Methods Appl. Mech. Eng. 186 (2000) 403–420.
[62] J.L. Marcelin, Genetic optimization of stiffened plates and shells, Internat. J. Numer. Methods Engrg. 51 (2001) 1079–1088.
[63] J.L. Marcelin, Genetic optimization of stiffened plates without the FE mesh support, Internat. J. Numer. Methods Engrg. 54 (2002) 685–694.
[64] M. Papadrakakis, N.D. Lagaros, Reliability-based structural optimization using neural networks and Monte Carlo simulation, Comput. Methods Appl. Mech. Eng. 191 (2002) 3491–3507.
[65] N.D. Lagaros, D.C. Charmpis, M. Papadrakakis, An adaptive neural network strategy for improving the computational performance of evolutionary structural optimization, Comput. Methods Appl. Mech. Eng. 194 (2005) 3374–3393.
[66] K.C. Giannakoglou, D.I. Papadimitriou, I.C. Kampolis, Aerodynamic shape design using evolutionary algorithms and new gradient-assisted metamodels, Comput. Methods Appl. Mech. Eng. 195 (2006) 6312–6329.
[67] N.D. Lagaros, A.Th. Garavelas, M. Papadrakakis, Innovative seismic design optimization with reliability constraints, Comput. Methods Appl. Mech. Eng. 198 (2008) 28–41.
[68] C. Majorana, S. Odorizzi, R. Vitaliani, Shortened quadrature rules for finite elements, Adv. Eng. Softw. 4 (1982) 52–57.
[69] J.M. Melenk, K. Gerdes, C. Schwab, Fully discrete hp-finite elements: fast quadrature, Comput. Methods Appl. Mech. Eng. 190 (2001) 4339–4364.
[70] A. Oishi, S. Yoshimura, Finite element analyses of dynamic problems using graphic hardware, Comput. Model. Eng. Sci. 25 (2) (2008) 115–131.
[71] C. Cecka, A.J. Lew, E. Darve, Assembly of finite element methods on graphics processors, Internat. J. Numer. Methods Engrg. 85 (2011) 640–669.
[72] K. Banas, F. Kruzel, J. Bielanski, Finite element numerical integration for first order approximations on multi- and many-core architectures, Comput. Methods Appl. Mech. Eng. 305 (2016) 827–848.
[73] S.E. Mousavi, H. Xiao, N. Sukumar, Generalized Gaussian quadrature rules on arbitrary polygons, Internat. J. Numer. Methods Engrg. 82 (2010) 99–113.
[74] W.K. Liu, Y. Guo, S. Tang, T. Belytschko, A multiple-quadrature eight-node hexahedral finite element for large deformation elastoplastic analysis, Comput. Methods Appl. Mech. Eng. 154 (1998) 69–132.
[75] P. Hansbo, A new approach to quadrature for finite elements incorporating hourglass control as a special case, Comput. Methods Appl. Mech. Eng. 158 (1998) 301–309.
[76] M.L. Bittencourt, T.G. Vanzquez, Tensor-based Gauss-Jacobi numerical integration for high-order mass and stiffness matrices, Internat. J. Numer. Methods Engrg. 79 (2009) 599–638.
[77] M. Kikuchi, Application of the symbolic mathematics system to the finite element program, Comput. Mech. 5 (1989) 41–47.
[78] G. Yagawa, G.-W. Ye, S. Yoshimura, A numerical integration scheme for finite element method based on symbolic manipulation, Internat. J. Numer. Methods Engrg. 29 (1990) 1539–1549.
[79] S. Rajendran, A technique to develop mesh-distortion immune finite elements, Comput. Methods Appl. Mech. Eng. 199 (2010) 1044–1063.
[80] T.J.R. Hughes, J.A. Cottrell, Y. Bazilevs, Isogeometric Analysis: CAD, finite elements, NURBS, exact geometry, and mesh refinement, Comput. Methods Appl. Mech. Eng. 194 (2005) 4135–4195.
[81] J.A. Cottrell, T.J.R. Hughes, Y. Bazilevs, Isogeometric Analysis, Wiley, 2009.
[82] R. Sevilla, S. Fernandez-Mendez, A. Huerta, NURBS-enhanced finite element method (NEFEM), Internat. J. Numer. Methods Engrg. 76 (2008) 56–83.
[83] R. Sevilla, S. Fernandez-Mendez, A. Huerta, 3D NURBS-enhanced finite element method (NEFEM), Internat. J. Numer. Methods Engrg. 88 (2011) 103–125.
[84] R. Sevilla, S. Fernandez-Mendez, Numerical integration over 2D NURBS-shaped domains with applications to NURBS-enhanced FEM, Finite Elem. Anal. Des. 47 (2011) 1209–1220.
[85] D. Schillinger, S.J. Hossain, T.J.R. Hughes, Reduced Bezier element quadrature rules for quadratic and cubic splines in isogeometric analysis, Comput. Methods Appl. Mech. Eng. 277 (2014) 1–45.
[86] P. Antolin, A. Buffa, F. Calabro, M. Martinelli, G. Sangalli, Efficient matrix computation for tensor-product isogeometric analysis: The use of sum factorization, Comput. Methods Appl. Mech. Eng. 285 (2015) 817–828.
[87] T.J.R. Hughes, A. Reali, G. Sangalli, Efficient quadrature for NURBS-based isogeometric analysis, Comput. Methods Appl. Mech. Eng. 199 (2010) 301–313.
[88] F. Auricchio, F. Calabro, T.J.R. Hughes, A. Reali, G. Sangalli, A simple algorithm for obtaining nearly optimal quadrature rules for NURBS-based isogeometric analysis, Comput. Methods Appl. Mech. Eng. 249–252 (2012) 15–27.
[89] R. Ait-Haddou, M. Barton, V.M. Calo, Explicit Gaussian quadrature rules for C1 cubic splines with symmetrically stretched knot sequences, J. Comput. Appl. Math. 290 (2015) 543–552.
[90] K.A. Johannessen, Optimal quadrature for univariate and tensor product splines, Comput. Methods Appl. Mech. Eng. 316 (2017) 84–99.
[91] M. Barton, V.M. Calo, Optimal quadrature rules for odd-degree spline spaces and their application to tensor-product-based isogeometric analysis, Comput. Methods Appl. Mech. Eng. 305 (2016) 217–240.
[92] A.P. Nagy, D.J. Benson, On the numerical integration of trimmed isogeometric elements, Comput. Methods Appl. Mech. Eng. 284 (2015) 165–185.
[93] H.-J. Kim, Y.-D. Seo, S.-K. Youn, Isogeometric analysis for trimmed CAD surfaces, Comput. Methods Appl. Mech. Eng. 198 (2009) 2982–2995.
[94] H.-J. Kim, Y.-D. Seo, S.-K. Youn, Isogeometric analysis with trimming technique for problems of arbitrary complex topology, Comput. Methods Appl. Mech. Eng. 199 (2010) 2796–2812.
[95] G.R. Liu, Mesh Free Methods, CRC Press, 2003.

[96] T. Belytschko, Y.Y. Lu, L. Gu, Element-free Galerkin methods, Internat. J. Numer. Methods Engrg. 37 (1994) 229–256.
[97] G. Yagawa, T. Yamada, Free mesh methods: a new meshless finite element method, Comput. Mech. 18 (1996) 383–386.
[98] G. Yagawa, T. Furukawa, Recent developments of free mesh method, Internat. J. Numer. Methods Engrg. 47 (2000) 1419–1443.
[99] K. Funahashi, On the approximate realization of continuous mappings by neural networks, Neural Netw. 2 (1989) 183–192.
[100] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (1989) 359–366.
[101] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[102] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (1986) 533–536.
[103] D.E. Rumelhart, J.L. McClelland, The PDP research group, in: Parallel Distributed Processing: Explolations in the Microstructure of Cognition, Volume 1: Foundation, MIT Press, 1986.
[104] G.E. Hinton, S. Osindero, Y. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (2006) 1527–1544.
[105] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy layer-wise training of deep networks, in: Proceedings of NIPS, 2006.
[106] Y. LeCun, Y. Bengio, G.E. Hinton, Deep learning, Nature 521 (2015) 436–444.
[107] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (2014) 1929–1958.
[108] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, Y. Bengio, Maxout network, J. Mach. Learn. Res. W&CP 28 (3) (2013) 1319–1327.
[109] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (2011) 2121–2159.
[110] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous distributed systems, 2016, arXiv: 1603.04467.
[111] S. Tokui, K. Oono, S. Hido, J. Clayton, Chainer: a next-generation open source framework for deep learning, in: Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems, NIPS, 2015.
[112] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan, Deep learning with limited numerical precision, in: Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 2015.
[113] M. Courbariaux, J.-P. David, Y. Bengio, Low precision storage for deep learning, 2014, arXiv:1412.7024.
[114] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1, 2016, arXiv:1602.02830.
[115] M.B. Gokhale, P.S. Graham, Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays, Springer, 2005.
[116] N. Srebro, A. Sharaibman, Rank trace-norm and max norm, in: Proceedings of the 18th Annual Conference on Learning Theory, 2005, pp. 545-560.