Archives of Computational Methods in Engineering (2021) 28:3433–3456
https://doi.org/10.1007/s11831-020-09507-0

ORIGINAL PAPER

Finite Elements Using Neural Networks and a Posteriori Error

Atsuya Oishi[1] · **Genki Yagawa**[2,3]

Received: 21 March 2020 / Accepted: 30 September 2020 / Published online: 15 October 2020

Abstract
As the finite element method requires many nodes or elements to obtain accurate results, the adaptive finite element method has been developed to obtain better results with fewer nodes, where error information is used to refine the initial mesh adaptively. In contrast to this, we propose in this paper two new methods to directly derive accurate results by artificial neural networks using information about the errors of analysis results. One of the proposed methods employs error information obtained using a posteriori error estimator to predict accurate solutions from a single analysis with a coarse mesh. The other utilizes error information obtained from comparison between two analysis results: analysis results by using a coarse mesh and those by using the corresponding refined mesh. In both methods above, the artificial neural network is employed to predict accurate results at any target point in the analysis domain based on the error information around the point. These methods are successfully tested in two-dimensional stress analyses.

1 Introduction

One of the most important issues of the finite element analyses is the development of solution methods to achieve an accurate solution with high speed.

Under the above circumstance, several techniques have been studied in the last few decades. One is an attempt to improve the computational efficiency of large-scale analysis using multiple CPUs and accelerators, where various hardware such as supercomputers [2, 4, 40, 164], multi-core CPUs [75, 106] or GPUs (Graphics Processing Units) [23, 121, 129] are employed.

The other is an attempt to reduce the number of nodes while maintaining accuracy called the adaptive method [5, 6, 156], which is developed as a method for obtaining a highly accurate solution while suppressing the drastic increase of the number of total nodes. In the method, the analysis is first performed with relatively coarse mesh consisting of a small number of elements of large sizes and then with partially refined mesh consisting of a larger amount of small elements in areas where a posterior error estimation indicates poor accuracy. Mesh refinement is repeated in areas of poor accuracy until the required accuracy is achieved, and the total number of nodes is much smaller than that would be obtained by dividing the whole domain into small elements. This method has the advantage that a solution with high accuracy can be obtained in the desired area. Many theoretical and numerical studies have been conducted on the method [11, 16, 21, 22, 32, 110, 124, 125, 147], and researches on code development methods [30, 109, 140] and parallel processing [13, 44, 78] have been performed. Studied also are the development of mesh subdivision methods indispensable for adaptive finite element methods [20, 66, 111, 141, 151], which have been applied to various problems [17, 81, 82].

Though the adaptive finite element method is promising in many cases, the final number of elements becomes much larger than the initial one, causing significant increase in analysis time. Another disadvantage of the method is that mesh refinement, which is repeatedly performed, has difficulties for domains of complicated shape.

On the other hand, the remarkable progress of computer performance has promoted the development of the machine learning [12, 112]. Among others, the hierarchical neural networks [59] have been successfully applied in various fields of computational mechanics [161]. The deep learning,

✉ Atsuya Oishi
aoishi@tokushima-u.ac.jp

Genki Yagawa
yagawag@gmail.com

[1] Graduate School of Technology, Industrial and Social Sciences, Tokushima University, 2-1 Minami-Johsanjima, Tokushima 770-8506, Japan

[2] University of Tokyo, Tokyo, Japan

[3] Toyo University, Tokyo, Japan

which has been attracting attention in recent years, is a technique using the hierarchical neural network with many intermediate layers [48]. The hierarchical neural networks are able to simulate arbitrary nonlinear maps, and have been applied to such areas as the nondestructive evaluation and the optimization problem. The big issue, however, is that, when trying to increase the number of layers and scale in order to perform more complex mappings, the training often needs very long time or does not progress at all due to the vanishing gradients problem [59], which is the reason why only small-scale neural networks have been used in many applications. But, Hinton, et al. made it possible to train and to construct large-scale hierarchical neural networks with many hidden layers [60], dramatically increasing the ability to extract the relationships hidden in a large amount of data.

In the present study, we propose two new methods (the Method A and the Method B) to derive accurate simulation results by the deep learning using information about the errors of analysis results. The present paper consists of 5 sections; Sect. 1 is to introduce the paper, Sect. 2 gives an overview of the neural networks and the deep learning reviewing their applications to the finite element analysis. Sections 3 and 4 propose new finite element analysis methods based on the error information and the deep learning. Finally, Sect. 5 summarizes the paper.

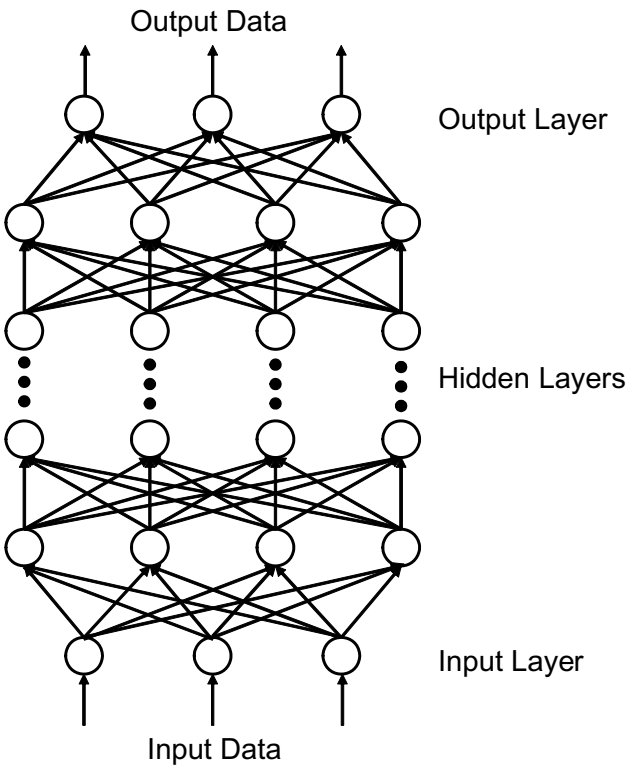

Fig. 1 Hierarchical neural network used in deep learning

2 Finite Elements Enhanced by Neural Networks and Deep Learning—A Review

2.1 Neural Networks and Deep Learning

Figure 1 shows the basic structure of a hierarchical neural network, which is the core technology of deep learning [59]. A hierarchical neural network has a structure, where the units, which are the basic components, are arranged hierarchically. When the total number of layers is N, they are divided into three categories: the input layer (the first layer), the hidden layers (from the second layer to the $(N-1)$ -th layer), and the output layer (the N-th layer) in order. In general, each unit in a layer of the fully connected network has a connection with units in adjacent layers, and each connection has a coupling coefficient called the weight. Each pair of units in the same layer has no connection and units located in nonadjacent layers have no connection either. Data input to the input layer propagates to the hidden layer through inter-unit connection, and further propagates from the hidden layers to the output layer through the inter-unit connection and is output from the output layer. It has been proved that hierarchical neural networks with three or more layers can approximate arbitrary continuous functions with arbitrary accuracy [38, 62], but the network structure and connection weights have to be determined through trial and error to achieve the desired approximation accuracy.

A unit is a mathematical model of a neuron with multiple inputs and single output, using the outputs of all the units in the lower layer as the input of the activation function. Each unit in the hidden layers and output layer takes as input the weighted sum of the outputs of all the units in the previous layer, and outputs the value processed by the corresponding activation function. A non-linear monotonically increasing function such as a sigmoid function [59] and a ReLU function [48] is used as the activation function.

A hierarchical neural network is capable of constructing a mapping from input data to output data (teacher signals) on a network by learning a large number of pairs of input data and output data. Pairs of input data and teacher signals are named training patterns. It is called the training to construct the mapping relation inherent in a lot of training patterns on the hierarchical neural network. Since the trained hierarchical neural network is given the mapping relationships, it outputs appropriate results even for inputs that are not provided in training, which is called the generalization.

Although the error back propagation learning method is the standard training method [59, 142], improvements have been made to training algorithms and network structures as applied to computational mechanics, including the training data selection method [25, 26], the method for preprocessing

of training data [167], the optimization of a neural network by deleting unnecessary units [166], the initial value determination method for connection weights [24], the comparative evaluation of several training methods for applications in computational mechanics [65, 134], the polynomial approximation based on Taylor expansion [132], the automatic determination of the number of units in intermediate layers [146], and the training method based on mutation operation [67].

When a hierarchical neural network has many hidden layers, the amount of calculation in training tends too large to complete the training in practically allowable time. This is partly because the revision of the connection weight in a layer far from the output layer becomes too small to train the network, which is called the vanishing gradients problem and solved by the prior layer-wise training using the Restricted Boltzmann Machine or the Auto-Encoder [8, 60]. Supported also by significant progress of computer, it has become possible to train large-scale hierarchical neural networks with many hidden layers.

The training of the multi-layered hierarchical neural networks is now called as the deep learning [48], which often uses very high-dimensional input data such as images and sounds together with the convolution layer [48]. There are several applications of the convolution layer in the field of computational mechanics, many of which employ microstructure images of composite material as input data [18, 29, 80, 96, 165]. Other applications of convolutional neural networks include the estimation of elastic heterogeneity and nonlinearity from the image of displacement field [131], the prediction of aerodynamic flow fields [10], and the eigenvalues are estimated from a large number of input data arranged in two dimensions like images [33].

Since the hierarchical neural networks and the deep learning have the ability to simulate various mapping relationships as described above, they can find rules inherent in various processes of finite element analysis by the following three steps,

(1) *Data preparation phase* In any field or process related to the finite element analyses, an input–output relationship is taken, setting the n-dimension vector $\{x_1^p, \ldots, x_n^p\}$ as the input and the m-dimensional vector $\{y_1^p, \ldots, y_m^p\}$ as the output, where p means the p-th data pairs out of a large set of data pairs and each component of the vector is of an integer or a real number.

(2) *Training phase* Using the input–output data pairs collected above, the training is started to find the mapping rules over them: a neural network is trained by taking the n-dimensional data $\{x_1^p, \ldots, x_n^p\}$ as the input to the network and the m-dimensional data $\{y_1^p, \ldots, y_m^p\}$ as the corresponding teacher signal. After the training finishes, we may have a multi-dimensional mapping $f()$ as

$$f : R^n \to R^m \tag{1}$$

where the mapping $f()$ achieved produces the input–output relationship as follows,

$$\left(y_1^p, \ldots, y_m^p\right) = f\left(x_1^p, \ldots, x_n^p\right) \tag{2}$$

(3) *Application phase* The original input–output relationship in the data preparation phase is replaced by the above mapping: Given a new input data, the corresponding output data is directly estimated with this mapping.

As for the selection of the input–output relationship in the data preparation phase, we can try any one irrespective of the size of input and output vectors, or whether it represents direct or inverse analysis [84]. The neural networks, however, are not omnipotent, then, for improper setting of input–output relationship or inadequate amount of data, they sometimes fail to learn a mapping accurate enough for the application.

Regarding the input–output data pairs in the data preparation phase, we can efficiently collect them by using the computational mechanics simulations. Experiment may be helpful, but it is hard to provide all the big data required for large-scale neural networks or deep learning only by experiment.

2.2 Two Categories of Finite Elements Enhanced by Neural Networks and Deep Learning

The hierarchical neural networks and the deep learning have been applied to many applications of the computational mechanics or the finite element method, which can be classified into two categories: One is the simple combination of the conventional finite element method with the neural networks or the deep learning, and the other is such application as the improvement of the capability of the finite element method itself by using the neural networks or the deep learning.

2.2.1 Simple Combination of FEM with Neural Networks or Deep Learning

The first category above includes the followings,

(1) Applications to nondestructive evaluation (defect identification, material property identification, structural identification, etc.), and
(2) Applications to optimization problems such as structural optimization.

The nondestructive evaluation is a typical inverse problem, where the neural networks are often employed [85]. Among the identifications of defects and damage are the applications to ultrasonic nondestructive evaluation [101, 118, 119], the defect identification of concrete structures by impact-echo method [133], the damage identification by vibration spectrum [170], the damage identification for substructure by vibration mode analysis [168], the damage identification of bridge structures [126], the identification of delamination in laminates [139], the damage identification of structures after earthquake [47], the damage identification of beams from statistical properties of dynamic response [97], the damage identification of structures using recurrent neural network [59, 71], and the damage assessment of composites using fuzzy neural network [136].

Applications to structural identification include the shape identification based on eigen frequencies [79], the model updating of T-beams from static and dynamic data [56], the identification of loads distributed across a cantilevered beam [19], and the prediction of proximal femur loads from bone morphology [41].

Other applications for identification and estimation include the estimation of Young's modulus and stresses of rocks from displacements at multiple locations using fuzzy neural networks [98], the estimation of residual stresses [153], the estimation of discharge capacity of radial-gated spillways [143], the prediction of load rating from structural parameters of bridges, which is equivalent to the prediction of dynamic characteristics of structures [55], the prediction of grain size in welded materials [35], the prediction of quality of welding [15], the contact pressure estimation of tire [28], the prediction of collision behavior of structures [88], the prediction of deformation of structures [34], the prediction of dam behaviors [144, 145], the estimation of maximum depth in deep drawing process [100], the damage prediction in extrusion process [51], and the femoral deformation prediction in multiscale analysis [52].

It is noted that the neural networks are often employed in the optimization problems in combination with one of the global optimization algorithms such as the genetic algorithms inspired by the biological evolution models [45, 108], the genetic programming [83, 84], the evolutionary algorithms or the evolutionary strategies [108], the probability-based Monte Carlo method [12, 112], the swarm intelligence algorithms [14, 130], where the neural networks help to improve the efficiency of individual evaluation that is repeatedly performed within the framework of the global search based on one of the algorithms above.

It is also known that the evaluation of individual by the finite element analysis that would require high computational load is replaced with a simplified model or surrogate model using a neural network, which can significantly improve the efficiency of the whole optimization process. In addition, the similar effect is achieved by reducing the number of evaluations of individuals, where the judgement of the suitability of each individual on various constraints is performed by the neural networks.

The above techniques are applied to structural optimization problems, including the truss structure optimization [171, 172], the frame structure optimization where neural networks are used to test whether some constraints are satisfied or not, and to reduce the number of time-consuming direct analyses in the Monte Carlo method [128], the structural optimization where the direct analysis for evaluation of each individual is replaced by a surrogate model using neural network to speed up the whole optimization process based on the evolutionary algorithm [86], the shape optimization of a flow path with the evolutionary algorithm [61], the structural optimization using neural networks trained to minimize both the error and other criteria [90], the design of structures optimized to resist earthquake [87], the surrogate models with the neural networks in structural optimization [46], the response surface using neural networks in structural optimization [70], the surrogate models using neural networks for deformation analysis in virtual reality environment [53], the tool shape optimization based on the response surface using neural network [150], the determination of shape parameters by neural networks for shapes classified using image moments [99], and the neural network-based surrogate model for carbon nanotubes with geometric nonlinearities [127].

Among applications to other optimization problems are the simplified model using neural networks for forward analysis in crack path control based on the particle swarm optimization [27], the surrogate neural networks for optimal control of tunnel construction [36], the surrogate neural networks for structural optimization of double grid structure for space structures [74], the surrogate neural networks for structural optimization of stub-girder structure [91], the response surface using a neural network for optimal design of automobile joints [113], the optimal design of automobile joints [114], and the clearance optimization for stamping process [54].

2.2.2 Finite Element Method Enhanced by Neural Networks

The second category is to apply the neural networks to a series of processes of the finite element analysis to improve the computational speed and the capability. Applications in this category include

(1) Constitutive equations of materials
(2) Solver
(3) Boundary condition
(4) Meshes (Element division and Domain decomposition)
(5) Contact search, and

(6) Element integration

Not a few studies on the applications of neural networks to material constitutive laws are performed, especially to nonlinear material models. This is because it is difficult to correctly express the nonlinearity and the path dependency with exact mathematical formulas, and the approximation capability of the neural networks in this respect has been recognized.

Applications to constitutive equations for a wide variety of nonlinear materials include the implicit constitutive equations that incorporate the concept of control theory [39], the constitutive equations that take into account the static recovery [63], the incremental constitutive equations [94], the velocity-dependent constitutive equation [68], the anisotropic material model [105], the constitutive equations for composite materials [92], the meso-model constitutive law in multiscale analysis [155], the application of neural networks trained by truncated-Newton method to viscoplastic constitutive equation [3], the estimation of stresses from strains using the support vector machine and a neural network [154], the rheological constitutive model using a recurrent neural network [115], the constitutive law of rubber materials [176], the estimation of stress–strain relationship from fuzzy representation of measurement error [37], the plastic hardening constitutive equation by Hopfield neural network [152], the autoprogressive method that gradually trains a neural network by repeating FEM analyses [43, 69], the stress–strain matrix represented in explicit form by multistage estimation using nested neural networks [57], the response surface for ceramic interface [138], the identification of parameters of repeated load history curves [93], the simplification of material models [149], and the application to homogenization analysis [89, 158]. Applications not to material constitutive equations but to similar structural properties include those to the moisture absorption and release characteristics that exhibit hysteresis characteristics [42] and to repetitive characteristics of structural joints [169].

Applications to solvers include an attempt to have neural networks represent the entire processes of the finite element analysis [95], the applications of neural networks to nonlinear analysis solvers [77, 148], the determination of analysis parameters by neural networks [123], the application to variational problems [102, 103], and the physics-informed neural networks [135]. There are also the direct solution method using interconnected neural networks and its implementation on parallel processors [122, 159, 162] and the solution of an elastoplastic problem using the Hopfield neural network [31].

As an example to boundary conditions, the non-reflection boundary conditions constructed on a hierarchical neural network are proposed [173].

Applications to element division and domain decomposition include the optimal node numbering for the Wavefront solver [73] and the applications of interconnected neural networks in combination with techniques based on the graph theory to domain decomposition for parallel processing [72].

As for the applications to contact analysis, studied is to calculate contact analysis parameters used in ANSYS with neural network [58]. Another application is to contact search [117, 120], which is an important process in dynamic contact analysis such as car crash simulation. In the node-segment type algorithm [9, 50] used in the contact-impact simulations, the problem of searching for the contact point between the master segment and the slave node is regarded as a mapping problem from the relative arrangement of the target master segment and the corresponding slave node to the local coordinate of the contact point on the master segment, and a hierarchical neural network is applied to simulate this mapping.

In order to reduce the computational cost for the element integration, various fast calculation methods are proposed [23, 104, 107].

While the numerical integration methods such as the Gauss–Legendre quadrature are often used for numerical quadrature of the element stiffness matrices, the accuracy of the integration varies depending on the shape of the element. To overcome this issue, researches have been done to obtain accurate results for elements of irregular shape using mathematical formula processing [76, 163]. As is well known, the accuracy of element integration is also improved by increasing the number of integration points. However, since the amount of computation is proportional to the number of integration points, it is desirable that the numerical integration should be performed with less integration points as possible with keeping the same level of accuracy. This issue is addressed with neural networks [116], where the optimal value of the numerical integration parameter in Gauss–Legendre integration is estimated for each element using neural network, and more accurate element integration is performed with a smaller number of integration points, or the number of integration points enough for the prescribed accuracy is estimated from the element shape.

3 Finite Elements Using Error Information and Deep Learning (Method A)

3.1 A Posteriori Error Estimation in Finite Elements

In order to estimate the analysis error from results obtained by using single mesh pattern in the finite element analysis, the a posteriori error estimation methods have been studied [1, 49, 156, 157]. Suppose that the basis function of the finite elements has the C^0 continuity at the element boundary,

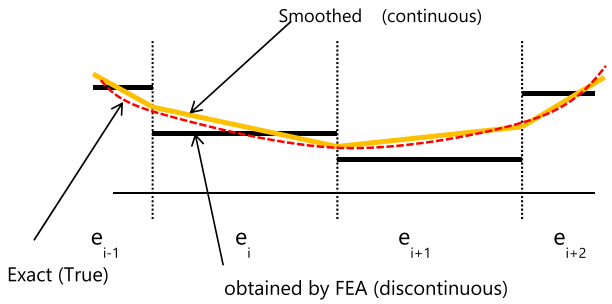

Fig. 2 Zienkiewicz-Zhu error estimator

where the stresses and the strains become discontinuous. But, the new stresses obtained by smoothing the discontinuous stresses above are known closer to the true values than the original ones, which is used by Zienkiewicz and Zhu to perform a posteriori error estimation, called the ZZ method [137, 175]. Figure 2 shows the schematic illustration of the ZZ method in one dimensional case. Since the stresses obtained by smoothing, shown by the piecewise linear solid lines, look closer to the true stresses shown as the dotted curve, the difference between the stresses obtained by the analysis and those by smoothing can be regarded as a good approximation of the error. The ZZ method is widely used as a posteriori error estimation method because it is easy to implement to the analysis code and has little computational overhead.

The adaptive finite element method is a technique to optimize element division based on a posteriori error estimation [5, 6, 156], in which, after the analysis with the initial coarse mesh, a posteriori error estimation is performed to improve the degree of approximation only in the region where relatively large error appears. The method is classified into three types: a method to refine the mesh locally (the h-adaptive method), a method to locally increase the order of the basis function (the p-adaptive method), and a method to rearrange the nodes locally (the r-adaptive method). Each can be performed independently, or combined. The h-adaptive method is the most commonly used, in which the mesh is locally refined where a posteriori error estimation predicts that the error of the analysis result is relatively large. This cycle of analysis, error estimation, and mesh refinement is repeated until the error is sufficiently small everywhere in the domain. Since the mesh refinement is performed not in whole domain but in some localized regions, the increase in the number of total nodes, which is directly related to analysis time, is suppressed. There is, however, a disadvantage that the total computational load is not necessarily low, because the analysis is repeated using the refined meshes.

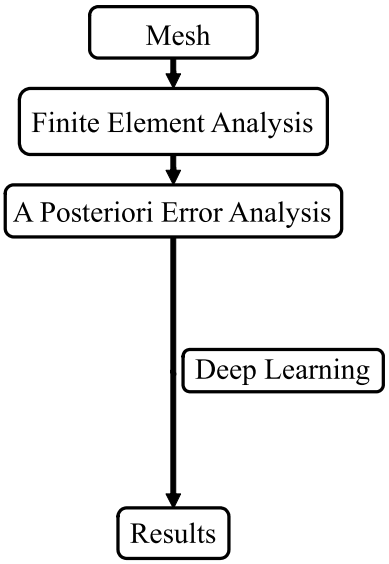

Fig. 3 Flowchart of Method A

3.2 Formulation of Method A

As described above, the error information is very useful to improve analysis accuracy. In addition, a posteriori error estimation, such as the ZZ method, has versatility that can be employed in various cases.

While the conventional adaptive finite element method performs mesh refinement adaptively based on error information to improve the accuracy of the solution, we propose here a new method called the Method A, employing a posteriori error estimation with the deep learning, where no mesh refinement nor reanalysis are needed. Figure 3 shows the flowchart of the proposed method, which consists of the following three processes:

Phase 1 Define such analysis parameters as shape of domain, load, boundary condition, etc. For each combination of them, we perform the finite element analysis with a relatively coarse mesh, a posteriori error estimation, and then the finite element analysis with a very fine mesh. Thus, a large number of data sets are collected, each of which consists of stress values of a target point and its surrounding points by the coarse mesh, the error information around the target point obtained by the coarse mesh, and the stress values of the target point calculated using the very fine mesh

Phase 2 A hierarchical neural network is trained using the data set prepared in Phase 1, which are Input data: stress values of a target point and its surrounding points by the coarse mesh and the error information of the stresses at these points, and Teacher

data: stress values of the target point calculated with the fine mesh. As a result, the hierarchical neural network is trained so that the analysis results of high accuracy are provided by inputting the analysis results and the error information with the coarse mesh,

Phase 3 The hierarchical neural network trained in Phase 2 above is applied to a new analysis problem that is not necessarily included in the training patterns. When the analysis result and the a posteriori error information calculated with the coarse mesh are input to the trained hierarchical neural network, the stress values of high accuracy that would be obtained with the fine mesh are estimated promptly

This method has significant versatility because it does not use any information on the boundary conditions, the shape of the analysis domain and the specific arrangement of nodes and elements, which will make it easier to apply the trained neural network to other situations. As for the speed of calculation, the estimation of stress values by the trained neural network is much faster than the usual finite element analysis with the fine mesh. In addition, the solution in the usual finite element analysis must be calculated for a mesh of the whole domain, whereas in the proposed method, the calculation is limited to the local area where stresses are to be estimated. On the other hand, Phases 1 and 2 above require a large amount of calculation, although these can be performed independently.

3.3 Examples

3.3.1 Problem Definition

The basic performance of the present method (Method A) is evaluated through simple examples as follows,

(1) *Analysis domain* Figure 4 shows the two-dimensional analysis domain. The shape of the analysis domain is a 4[m]×4[m] equilateral square with the bottom being fixed, and the material constants are assumed to be those of steel.

(2) *Element division* The whole domain is divided into equally-sized square quadrilateral linear element. Employed are two different meshes, the coarse mesh with 16 (4×4) elements and the fine mesh with 65,536 (256×256) elements.

(3) *Loading conditions* As shown in Fig. 4, the coarse mesh has 10 edges of 1 m long each at the side and the top, which are numbered as 1, 2 and so on. Equally distributed external loads (1 [N/m]) are applied at two edges selected from the ten edges above with the loading directions randomly set to one of 12 directions shown in Fig. 5. Considering 45 ways ($= {}_{10}C_2$) of edge selection and 144 ($= 12 \times 12$) ways of load direction per edge selection, the total number of loading conditions is 6480 ($= 45 \times 144$).

(4) *Element integration* Integral of each element is performed using the Gauss–Legendre quadrature with four integration points per element.

(5) *Stress evaluation points* The stresses are calculated for each of 6480 loading condition above using the coarse and the fine meshes, respectively. Figure 6 shows uniformly located 1600 ($=40\times40$) stress evaluation points. In each analysis using either the coarse or the

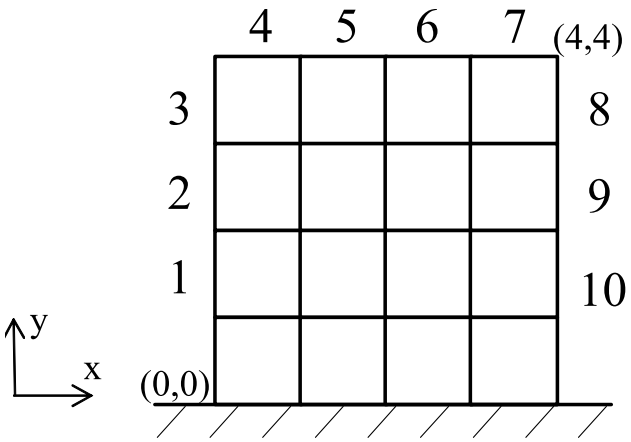

Fig. 4 Analysis domain

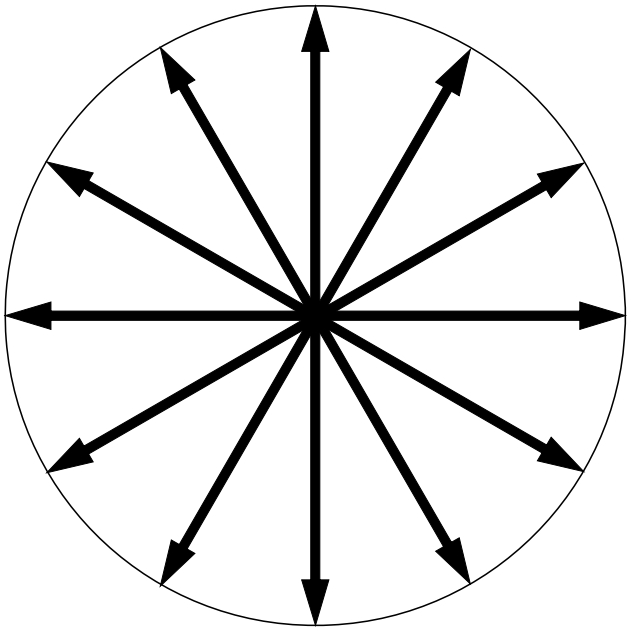

Fig. 5 Twelve directions of applied forces

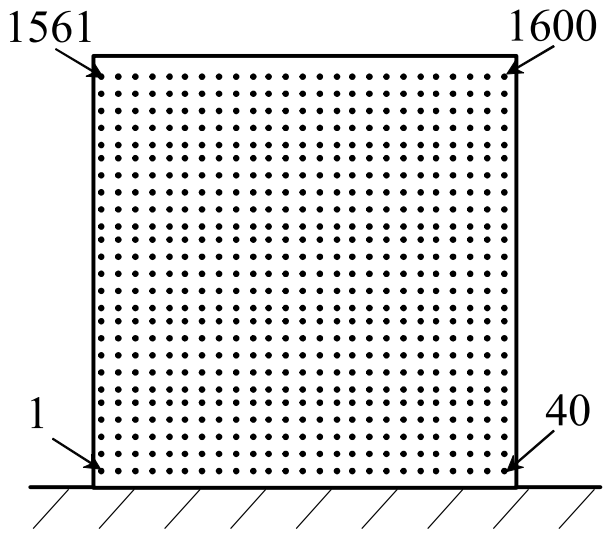

Fig. 6 Evaluation points

fine mesh, the stresses at these points are calculated as $(\sigma_x^C, \sigma_y^C, \tau_{xy}^C)$ and $(\sigma_x^F, \sigma_y^F, \tau_{xy}^F)$, respectively.

(6) *Stress smoothing* In the analysis using the coarse mesh, the smoothed stresses $(\sigma_x^S, \sigma_y^S, \tau_{xy}^S)$ at the stress evaluation point are calculated based on the ZZ method. Firstly, the smoothed stresses at each node are calculated by the simple averaging operation, which are then employed to calculate another smoothed stresses at a stress evaluation point in an element with the shape functions.

Regarding the stress values of the stress evaluation point group ($y = 3.85$) under the load condition shown in red in Fig. 7, Fig. 8 shows the stress value versus the x coordinate in Fig. 4, where σ_x (coarse) is achieved by the coarse mesh of 16 elements, σ_x (smoothing) by smoothing the σ_x (coarse), and σ_x (fine) by the fine mesh of 65,536 elements. Figures 9 and 10 are the same as above but for σ_y and τ_{xy}, respectively.

Figures 11 and 12, respectively, show how the stress values at positions (x = 1.15) and (x = 3.85) in the stress evaluation point group (y = 3.85) in Fig. 7 converge according with the increase of the number of elements used for analyses, where the stress values obtained by the analyses with seven types of meshes of 16, 64, 256, 1024, 4096, 16,384 and 65,536 elements are given.

3.3.2 Training Patterns for Deep Learning

A training pattern for the deep learning consists of input data to hierarchical neural networks and the corresponding teacher signals. The input data in this study are generated from the stress values at five stress evaluation points around

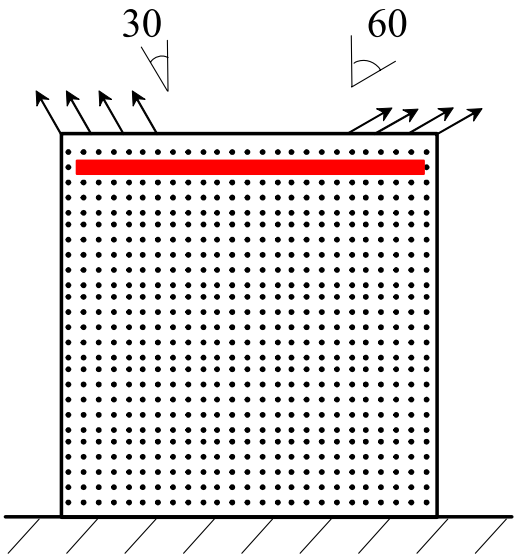

Fig. 7 Test problem

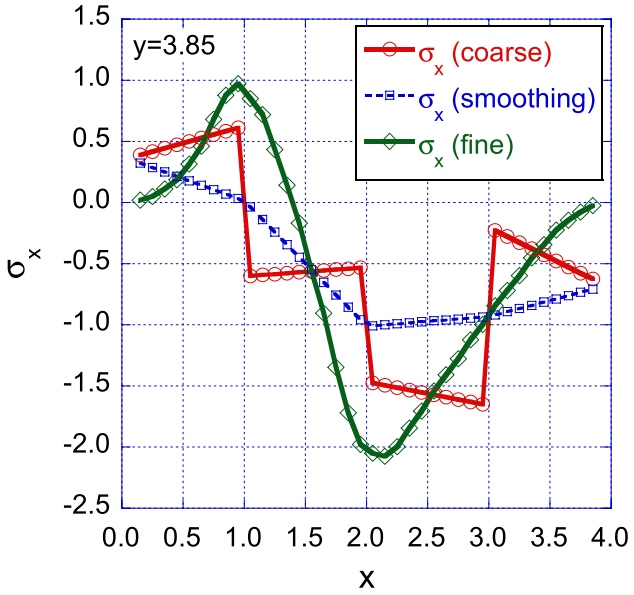

Fig. 8 Calculated value of σ_x at y = 3.85

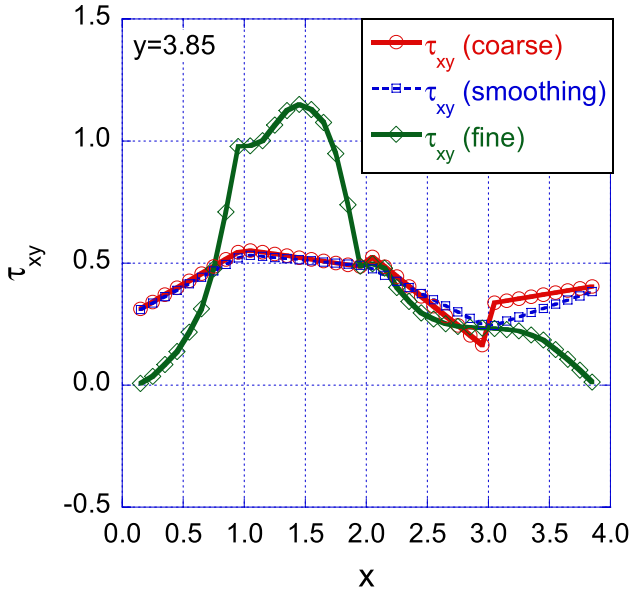

Fig. 10 Calculated value of τ_{xy} at y = 3.85

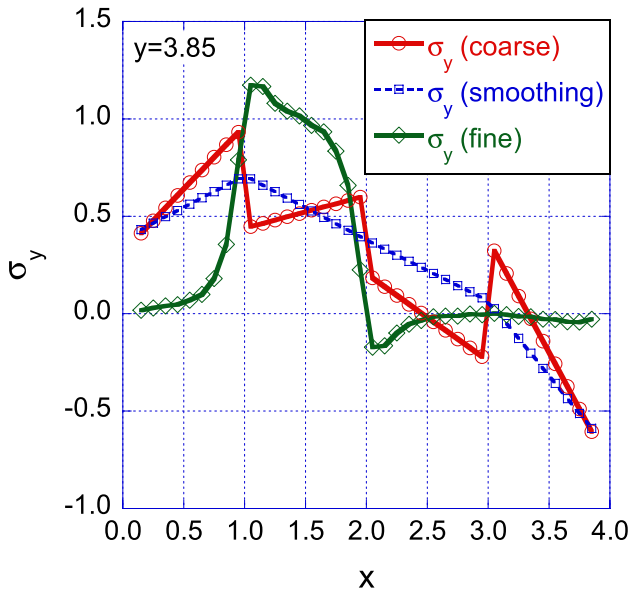

Fig. 9 Calculated value of σ_y at y = 3.85

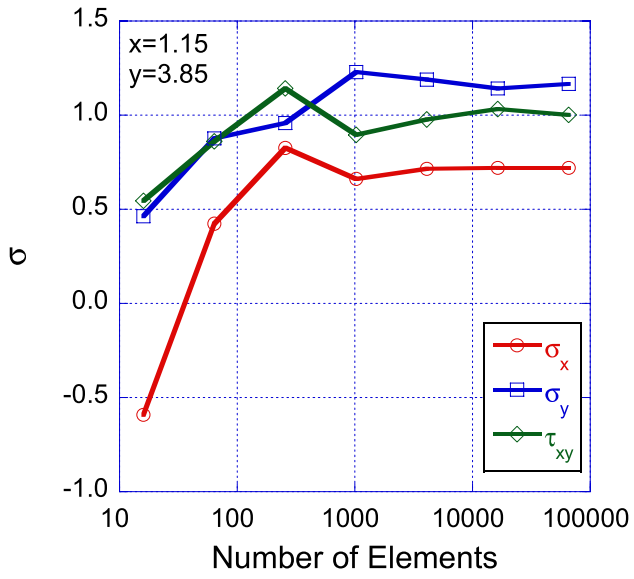

Fig. 11 Stresses at $(x, y) = (1.15, 3.85)$ versus number of elements

a target point shown in Fig. 13. Taking an arbitrary stress evaluation point as a point of interest (P_A), its four surrounding points ($P_{A1}, P_{A2}, P_{A3}, P_{A4}$) are selected. Both the stresses ($\sigma_x^C, \sigma_y^C, \tau_{xy}^C$) and the smoothed stresses ($\sigma_x^S, \sigma_y^S, \tau_{xy}^S$) at these five points obtained using the coarse mesh of 16 elements are used to generate twenty-seven input data for the neural network. Twelve data among the twenty-seven input data consist of the difference between stresses ($\sigma_x^C, \sigma_y^C, \tau_{xy}^C$) at P_A and those at each of ($P_{A1}, P_{A2}, P_{A3}, P_{A4}$), whereas the other fifteen data the difference between ($\sigma_x^C, \sigma_y^C, \tau_{xy}^C$) and ($\sigma_x^S, \sigma_y^S, \tau_{xy}^S$) at each of these five points.

On the other hand, the corresponding teacher signals are set to be the differences between the stress values ($\sigma_x^F, \sigma_y^F, \tau_{xy}^F$) and ($\sigma_x^C, \sigma_y^C, \tau_{xy}^C$) at the target point.

Tables 1 and 2 summarize the input data and the teacher signals, respectively, where ${}^{P_A}\sigma_x^S$ means σ_x^S at P_A, for example. Note that all the input data are generated only from values obtained with a coarse mesh, while the teacher signals include analysis results with the corresponding fine mesh.

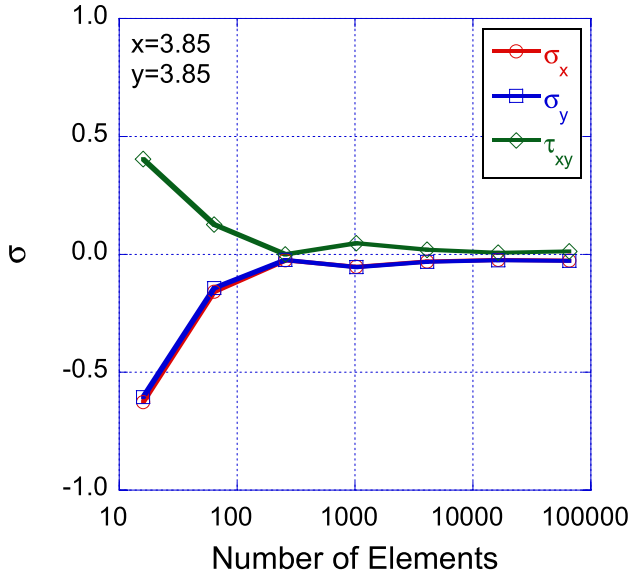

Fig. 12 Stresses at $(x, y) = (3.85, 3.85)$ versus number of elements

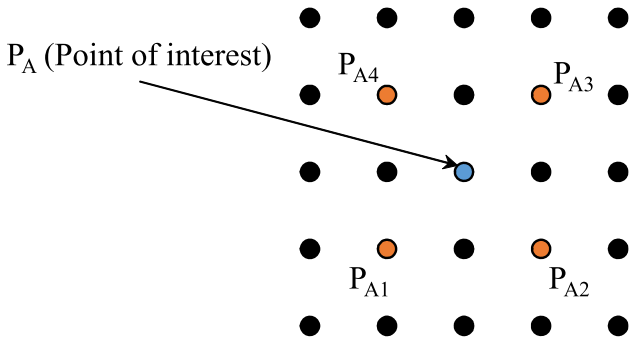

Fig. 13 Input data for neural networks

Table 1 Input data for deep learning (Method A)

$P_A\sigma_x^S - P_A\sigma_x^C$	$P_{A1}\sigma_x^S - P_{A1}\sigma_x^C$ $P_{A2}\sigma_x^S - P_{A2}\sigma_x^C$ $P_{A3}\sigma_x^S - P_{A3}\sigma_x^C$ $P_{A4}\sigma_x^S - P_{A4}\sigma_x^C$	
$P_A\sigma_y^S - P_A\sigma_y^C$	$P_{A1}\sigma_y^S - P_{A1}\sigma_y^C$ $P_{A2}\sigma_y^S - P_{A2}\sigma_y^C$ $P_{A3}\sigma_y^S - P_{A3}\sigma_y^C$ $P_{A4}\sigma_y^S - P_{A4}\sigma_y^C$	
$P_A\tau_{xy}^S - P_A\tau_{xy}^C$	$P_{A1}\tau_{xy}^S - P_{A1}\tau_{xy}^C$ $P_{A2}\tau_{xy}^S - P_{A2}\tau_{xy}^C$ $P_{A3}\tau_{xy}^S - P_{A3}\tau_{xy}^C$ $P_{A4}\tau_{xy}^S - P_{A4}\tau_{xy}^C$	
$P_{A1}\sigma_x^C - P_A\sigma_x^C$	$P_{A2}\sigma_x^C - P_A\sigma_x^C$ $P_{A3}\sigma_x^C - P_A\sigma_x^C$ $P_{A4}\sigma_x^C - P_A\sigma_x^C$	
$P_{A1}\sigma_y^C - P_A\sigma_y^C$	$P_{A2}\sigma_y^C - P_A\sigma_y^C$ $P_{A3}\sigma_y^C - P_A\sigma_y^C$ $P_{A4}\sigma_y^C - P_A\sigma_y^C$	
$P_{A1}\tau_{xy}^C - P_A\tau_{xy}^C$	$P_{A2}\tau_{xy}^C - P_A\tau_{xy}^C$ $P_{A3}\tau_{xy}^C - P_A\tau_{xy}^C$ $P_{A4}\tau_{xy}^C - P_A\tau_{xy}^C$	

Table 2 Teacher signals for deep learning (Method A)

$P_A\sigma_x^F - P_A\sigma_x^C$	$P_A\sigma_y^F - P_A\sigma_y^C$	$P_A\tau_{xy}^F - P_A\tau_{xy}^C$

Due to the constraint that the stress values around the point of interest are to be employed, 1444 points are selected as the target point of interest among the 1600 stress evaluation points excluding the points at the outermost circumference. For each target point, a training pattern is generated for each load condition, which results in about 9.36 million ($= 6480 \times 1444$) patterns in total. Note that each data of the training pattern is normalized to the range of [0, 1].

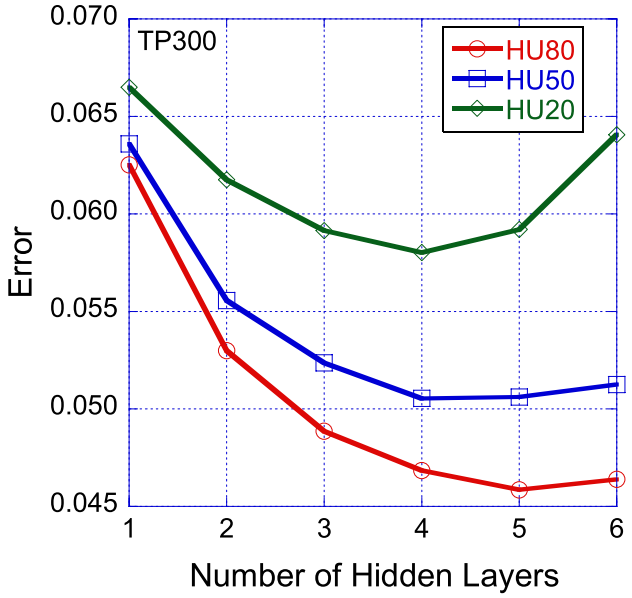

Fig. 14 Errors versus number of hidden layers for various number of units (Method A)

3.3.3 Deep Learning

In order to train the hierarchical neural network, we randomly choose two groups from about 9.36 million patterns prepared above: the patterns for training and the test patterns to verify the generalization capability of the trained neural network. The number of training patterns is varied as 300,000, 200,000, 100,000 and 50,000, while the number of test pattern is fixed as 100,000 with the same patterns in all the cases. The hierarchical neural network used in this study has 27 units in the input layer and 3 units in the output layer, the number of units in each hidden layer is varied as 20, 50, and 80, and the number of hidden layers as 1 to 6, and the number of training epochs is fixed to 10,000.

Figure 14 shows how the errors in the stress estimation vary with the number of units in each hidden layer and the number of hidden layers, when using 300,000 training patterns. In the figure, the vertical axis shows the average error for 100,000 test patterns, the horizontal axis the number of hidden layers, HU20, HU50, and HU80 mean 20, 50, and 80 units in each hidden layer, respectively, and TP300 the number of training patterns, which is 300,000. The error is defined as

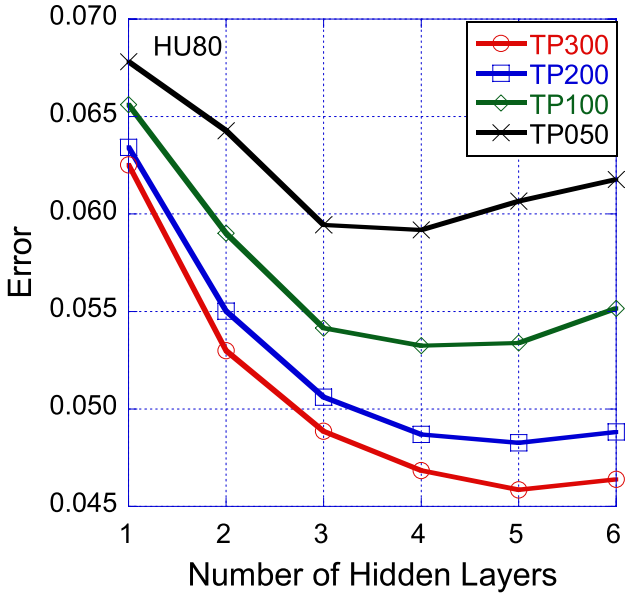

Fig. 15 Errors versus number of hidden layers for various number of training patterns (Method A)

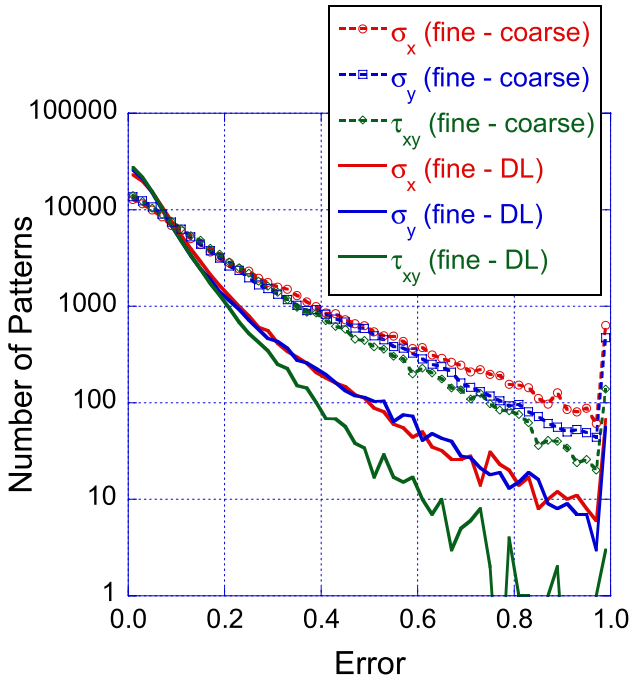

Fig. 16 Number of test patterns versus error (Method A)

$$\text{Error} = \frac{1}{N_{TP}} \sum_{i=1}^{N_{TP}} \sum_{j=1}^{N_{OU}} \left| O_j^i - T_j^i \right| \tag{3}$$

where N_{TP} is the number of test patterns, N_{OU} the number of units in the output layer of the neural network,O_j^i the output value of the j-th output unit in the i-th test pattern, and T_j^i the corresponding teacher signal. It can be seen from Fig. 14 that the smallest error is obtained when the number of hidden layers is 5 and the number of units in each hidden layer is 80.

Figure 15 shows the influence of the number of training patterns on the estimation error (Eq. 3) for the test patterns with the number of units in each hidden layer fixed to 80, where TP300, TP200, TP100 and TP050 mean the number of training patterns being 300,000, 200,000 and 100,000, and 50,000, respectively, and the vertical axis is the estimation error for the test patterns. It can be seen from the figure that the error for the test patterns decreases as the number of training patterns increases. In view of the above results, the hierarchical neural network with 5 hidden layers and 80 units in each hidden layer is chosen as the best one for the stress estimation, which is further trained up to the number of training epochs of 30,000 with 300,000 training patterns. Then, the error for the test pattern is reduced from 0.04586 with 10,000 training epochs to 0.04294 with 30,000 training epochs.

3.3.4 Accuracy of Estimated Stress

The accuracy of the stress estimated by the proposed method (Method A) is studied for the best-trained neural network: 5 hidden layers, 80 units in each hidden layer and 30,000 epochs of training.

Figure 16 shows the distribution of errors for the stress estimated with 100,000 test patterns, where the horizontal axis is the absolute value of the difference between two stress values: one estimated by the deep learning with the stresses and the smoothed ones obtained using the coarse mesh of 16 elements and the other using the fine mesh of 65,536 elements, and the vertical axis the number of patterns. For comparison, the distribution of the absolute value of the difference between the stress value obtained by the fine mesh of 65,536 elements and that obtained by the coarse mesh of 16 elements is indicated by dotted line. The figure shows that the patterns of the error distribution of stresses estimated by the deep learning concentrate nearer to 0.0 when compared to those obtained by using the coarse mesh of 16 elements

In the following, we estimate the stresses under the condition shown in Fig. 7 by the trained neural network, where three different groups of evaluation points are selected: 38 points in the top (y = 3.85) shown in red in Fig. 7, 38 points in the center (y = 1.95) and 38 points in the bottom (y = 0.15). The estimated results of σ_x in these three groups are shown in Figs. 17, 18 and 19, respectively. Similarly, the estimated results of σ_y in the above three groups are shown in Figs. 20, 21 and 22, respectively, and those of τ_{xy} in Figs. 23,

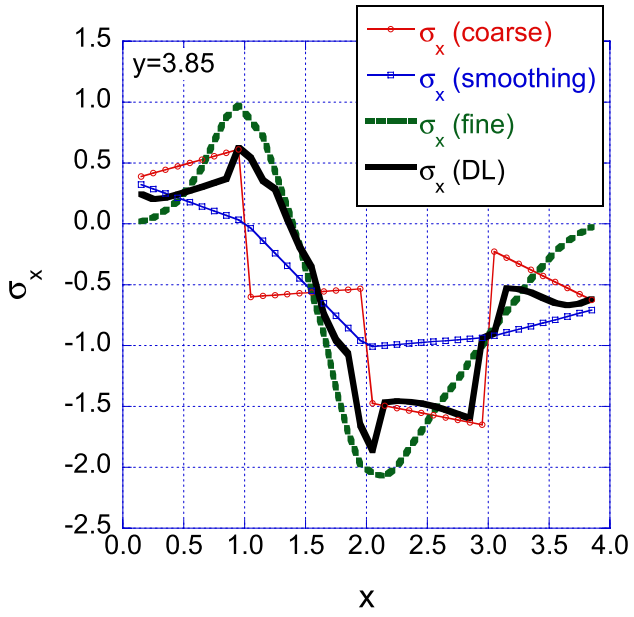

Fig. 17 Estimated value of σ_x at y = 3.85 (Method A)

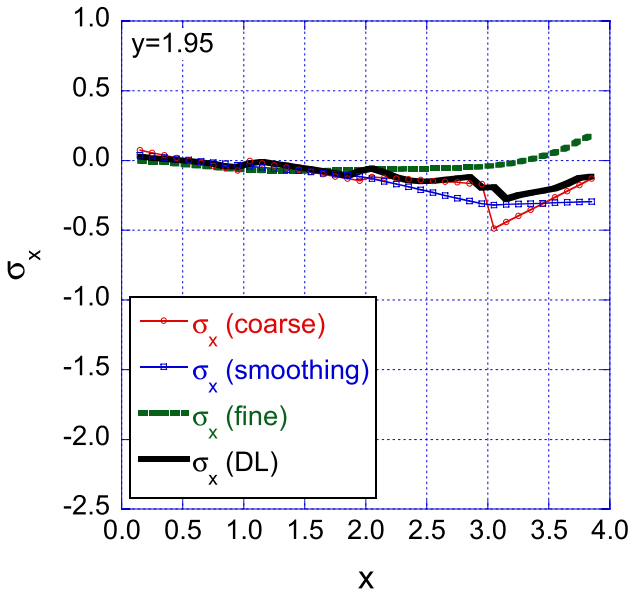

Fig. 18 Estimated value of σ_x at y = 1.95 (Method A)

24 and 25, respectively. For comparison, the stresses calculated by using a fine mesh of 65,536 elements (the reference values), and those by using a coarse mesh of 16 elements and its smoothed ones are also shown. It is noted that the stresses estimated by the deep learning above are close to the reference ones, showing the high generalization capability of the trained neural network given in this study.

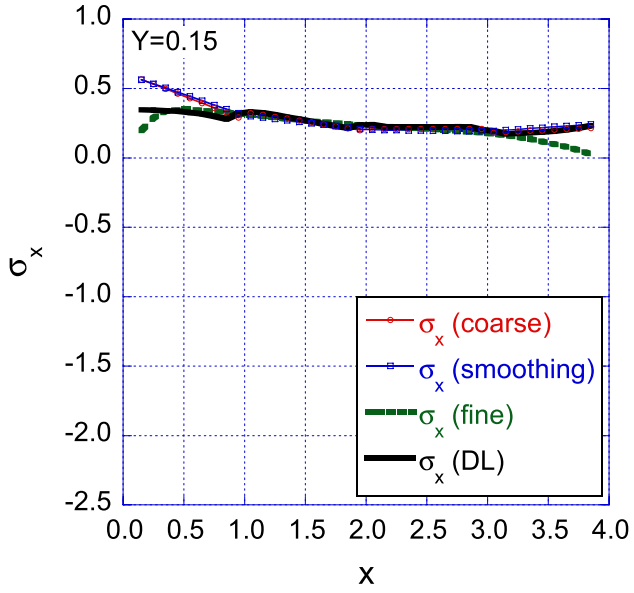

Fig. 19 Estimated value of σ_x at y = 0.15 (Method A)

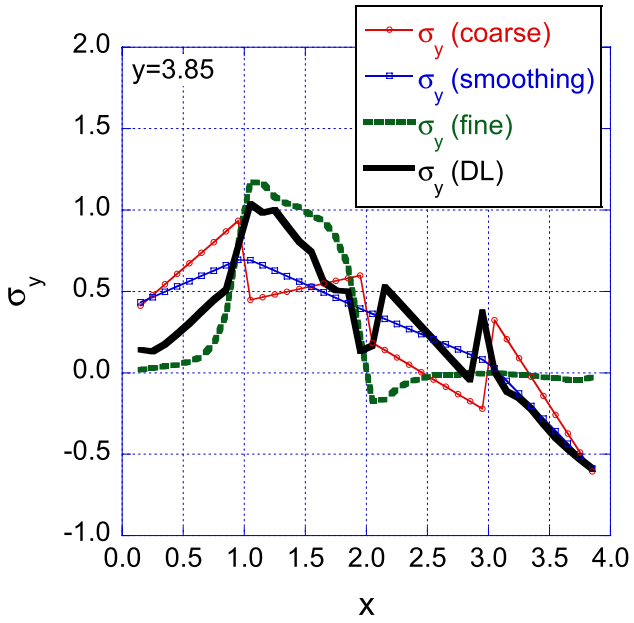

Fig. 20 Estimated value of σ_y at y = 3.85 (Method A)

3.3.5 Computational Load and Cost Performance

We discuss here the CPU time required in each phase of Sect. 3.2: Measured in the same environment (CPU: AMD Ryzen 2700X, Memory: 16 GB, OS: CentOS 7).

Figure 26 shows the CPU time required for stress estimation by the trained neural network, where the horizontal axis is the number of hidden layers in the hierarchical neural

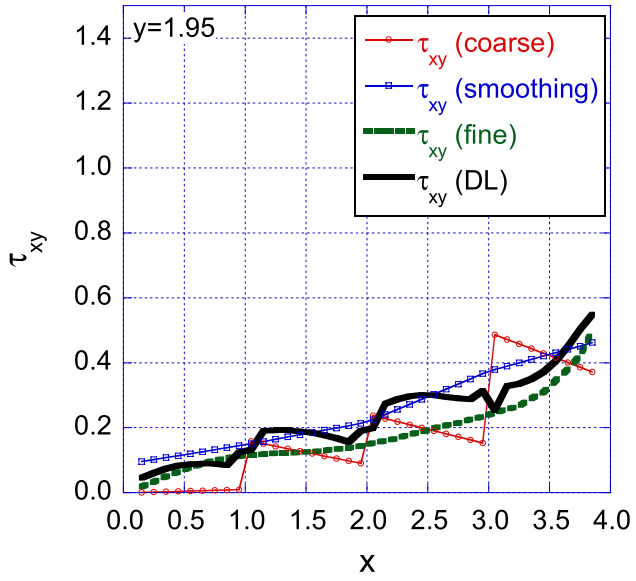

Fig. 21 Estimated value of σ_y at y = 1.95 (Method A)

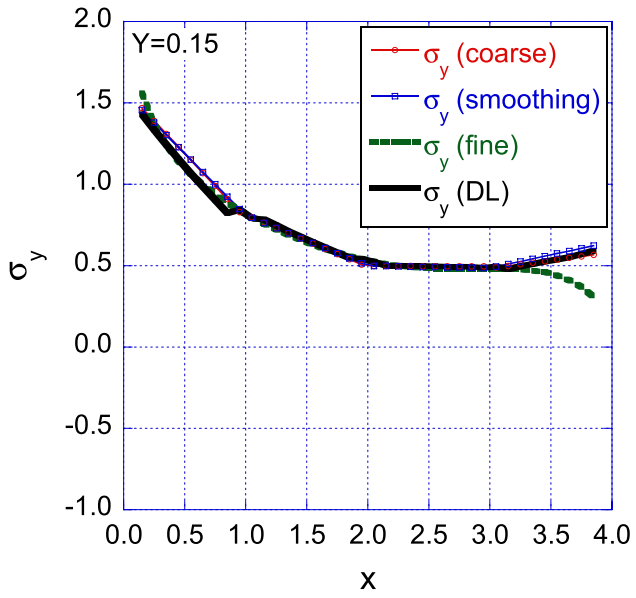

Fig. 22 Estimated value of σ_y at y = 0.15 (Method A)

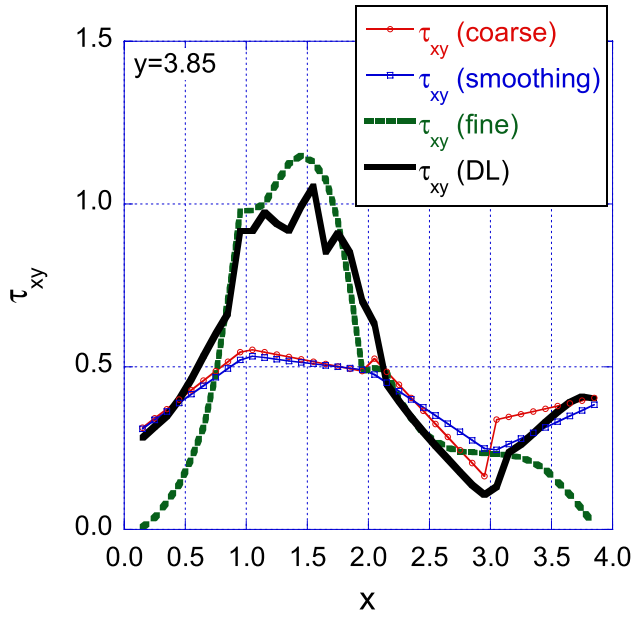

Fig. 23 Estimated value of τ_{xy} at y = 3.85 (Method A)

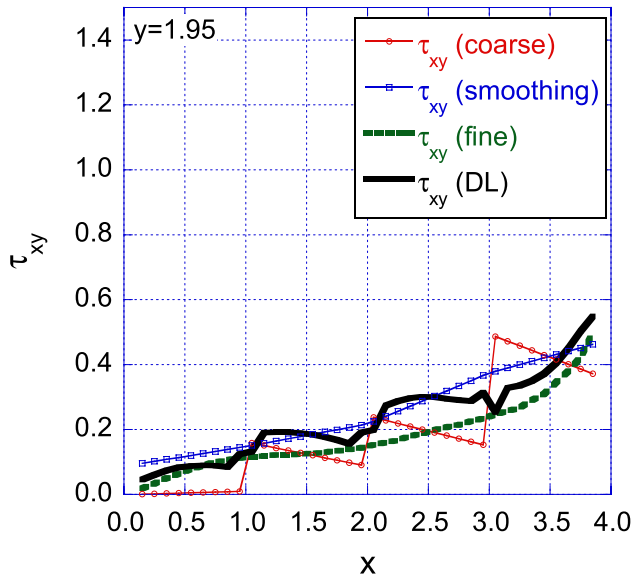

Fig. 24 Estimated value of τ_{xy} at y = 1.95 (Method A)

network, and the vertical axis the CPU time (in milliseconds) required for stress estimation for each location, varying the number of hidden layers and the number of units per hidden layer. It is shown in the figure that the neural network used here (five hidden layers with 80 units per layer) requires the CPU time for estimation of about 0.03 ms and if a slightly larger neural network is used, it does about 0.1 ms.

As shown in Fig. 27, where the horizontal axis is the number of elements and the vertical axis the CPU time required for each case, the CPU times of the finite element analysis used in the above example are, respectively, 0.05 ms, 0.25 ms, and 40 s or more with the mesh of 16 elements, that of 64 elements, and that of 65,536 elements used for teacher signals, suggesting that the required CPU time increases rapidly with the analysis scale.

On the other hand, the CPU time for inference by the trained neural networks increases rather slowly against the network size as shown in Fig. 26. From the viewpoint of CPU time, the superiority of the present method will be much higher in the three-dimensional cases than in the two-dimensional cases, and the CPU time required for inference

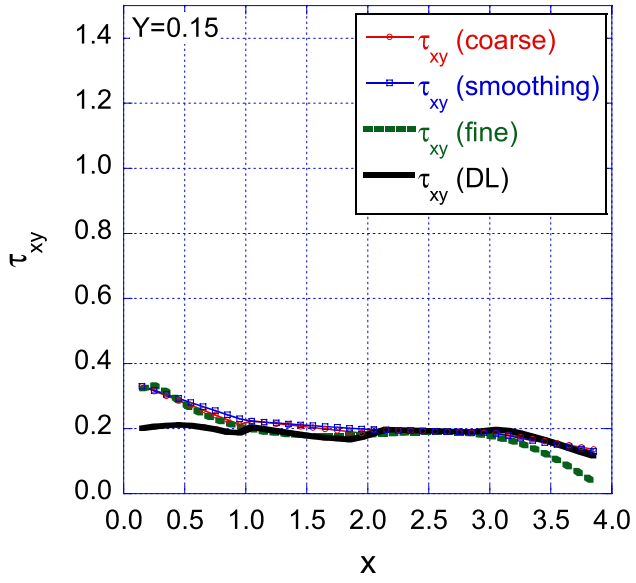

Fig. 25 Estimated value of τ_{xy} at y = 0.15 (Method A)

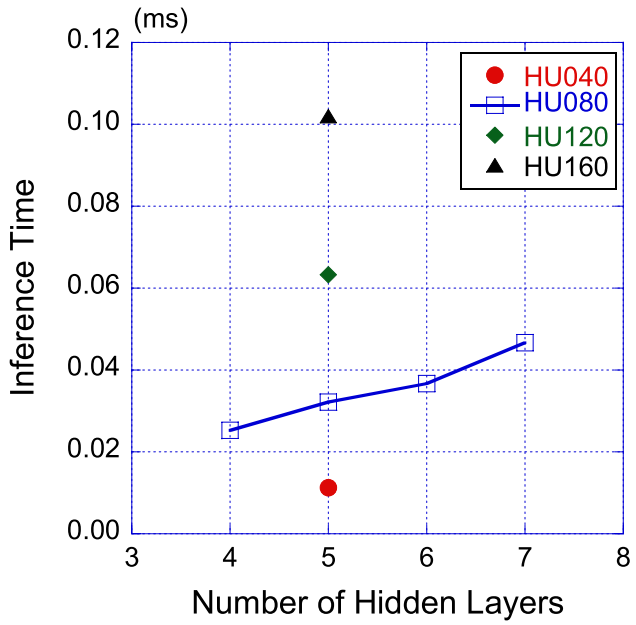

Fig. 26 Inference time versus number of hidden layers for various number of units in Phase 3

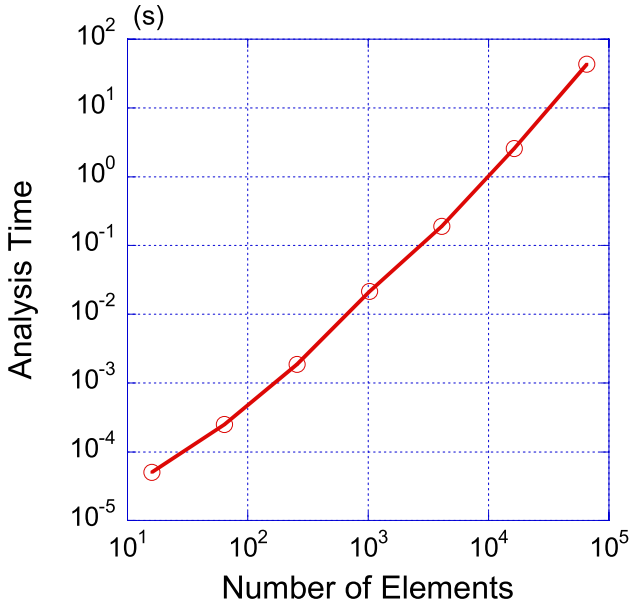

Fig. 27 Analysis time versus number of elements in Phase 1

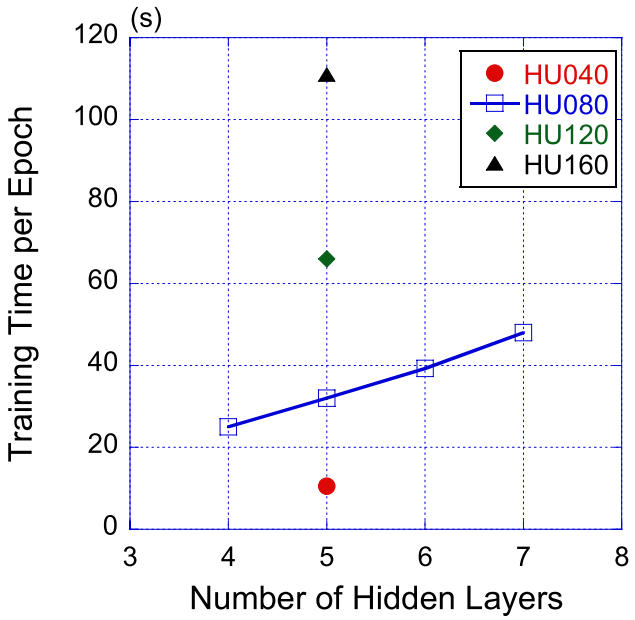

Fig. 28 Training time versus number of hidden layers for various number of units in Phase 2

can be significantly shortened by using the ReLU activation function and also by using low-precision computing units [116].

Finally, we discuss the CPU time for the deep learning in Phase 2. Figure 28 shows the CPU times required for each training epoch using 300,000 training patterns, varying the number of hidden layers and that of units per hidden layers, where the horizontal axis is the number of hidden layers. Under the conditions used in the example (five hidden layers, 80 units per hidden layer), the CPU time required for each training epoch is about 32 s, i.e. 270 h for 30,000 training epochs.

Although the training process requires a huge amount of CPU time as discussed above, it is possible to speed up the training process significantly by using a dedicated computing devices such as the GPUs suitable for the deep learning,

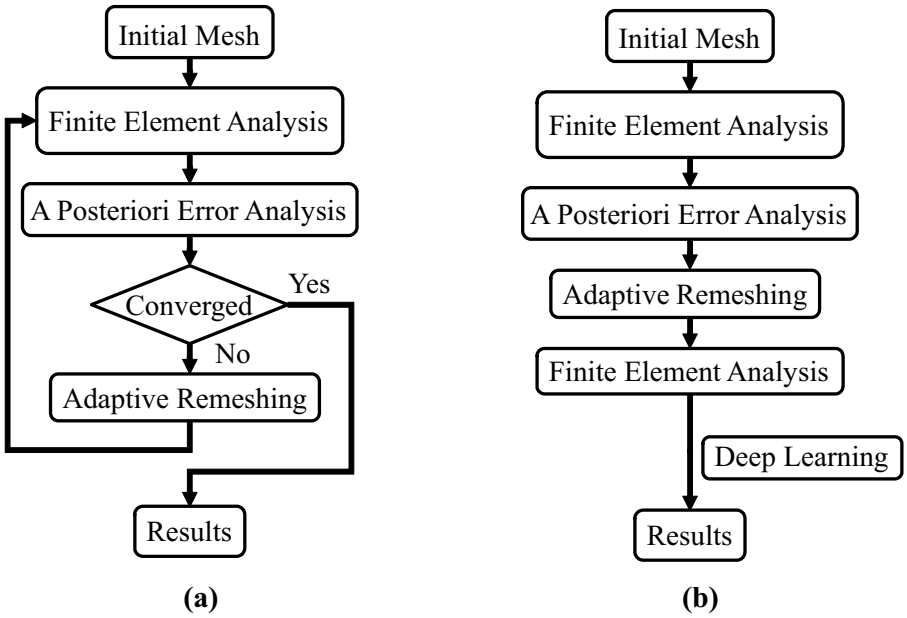

Fig. 29 Flowcharts: **a** Standard adaptive method, **b** Method B

since the process is independent of the application phase (Phase 3).

4 Adaptive Finite Element Analysis Using Deep Learning (Method B)

4.1 Asymptotic Behavior of Errors in the Finite Element Method

It is well known that a better solution is achieved as a finer mesh is employed [7, 64, 174]. The behaviors of analysis error changing an element size have been studied, and the knowledge obtained from these studies are useful to improve the solution method. As an example, a method is developed to predict the stress intensity factor at the crack tip with high accuracy based on the relationship between the density of nodes and the solution error [160], where the true stress is assumed to be predicted using the equation as follows,

$$\sigma_{\mathrm{FEM}} = \sigma_{\mathrm{EXACT}} + aN^{-\delta} \tag{4}$$

where σ_{FEM} is the stress obtained by the finite element analysis, σ_{EXACT} the true stress, a and δ are constants, and N the density of nodes. By estimating the constants from multiple analyses varying the density of nodes, we can determine σ_{EXACT} using Eq. (4).

4.2 Formulation of Method B

As described above, very useful information for the improvement of the analysis results is obtained using the asymptotic property of the error with multiple analysis results of different element sizes.

Based on this, we propose here a method to improve the adaptive finite element method (h-adaptive method), in which we perform mesh refinement repeatedly based on error information to improve the accuracy of solution.

On the other hand, accurate (converged) solutions are directly estimated here based on the asymptotic behavior of errors measured in two solutions of different meshes: solutions with initial mesh and those with refined mesh after the adaptive mesh refinement in the h-adaptive method.

Figure 29 shows the flowcharts of the conventional adaptive finite element method versus the present method, where Fig. 29a illustrates the conventional adaptive method that repeats the cycle consisting of analysis, a posteriori error estimation and mesh refinement, while Fig. 29b the proposed method (Method B), in which the cycle is performed only once and deep learning is used to estimate the accurate (converged) solutions from two analysis results of initial and refined meshes. A hierarchical neural network used in the deep learning is trained so that it outputs accurate (converged) results using analysis results of two meshes as input.

The method consists of the following three phases:

Phase 1 Define a problem to be analyzed with various parameters as shape of analysis domain, external

load conditions, fixed displacement conditions, etc. For various combinations of each parameter, the finite element analyses are performed with three meshes of different fineness: a coarse mesh, a mesh generated by refining the coarse mesh, and a very fine mesh. As a result, a large number of data sets using three different meshes are collected

Input data: Stress values at a point of interest and points surrounding the point, both calculated using the coarse and its refined meshes.

Teacher data: Stress values at a point of interest calculated using a very fine mesh.

It is expected that the trained hierarchical neural network predicts the stresses equivalent to those of a very fine mesh when the stresses of the coarse and its refined meshes are given.

Phase 2 A hierarchical neural network is trained using the data set prepared in Phase 1 with the input and teacher data, respectively, as

Phase 3 The hierarchical neural network constructed in Phase 2 is applied to any problem. When two analysis results obtained using the initial coarse and its refined meshes in the adaptive finite element method, respectively, are given, the trained hierarchical neural network promptly outputs the stress values that would be obtained using a very fine mesh.

4.3 Examples

4.3.1 Problem Definition

The basic performance of the hierarchical neural network for stress estimation, which is the core of the present method (Method B), is tested through some analyses using the same problems as in Sect. 3.3.

In each analysis using the coarse mesh (16 elements), the refined mesh (64 elements), and the very fine mesh (65,536 elements), the stress values at each node in an element are first calculated from those at the integration points in the element using the shape functions, and then the stress values at each stress evaluation points (shown in Fig. 6) in the element are calculated from the stress values at nodes of the element using the shape functions. Finally, for each analysis, we get stress values at 1600 (=40×40) stress evaluation points evenly allocated in the analysis domain with the coarse mesh ($\sigma_x^C, \sigma_y^C, \tau_{xy}^C$), the refined mesh ($\sigma_x^M, \sigma_y^M, \tau_{xy}^M$) and the very fine mesh ($\sigma_x^F, \sigma_y^F, \tau_{xy}^F$), respectively.

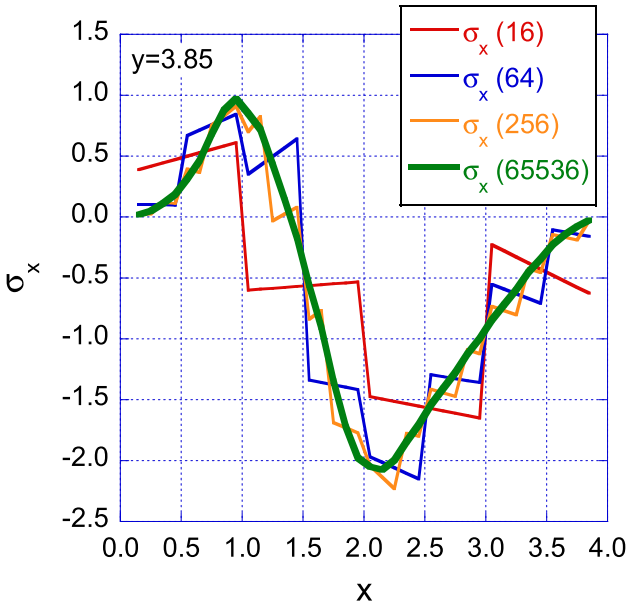

Fig. 30 Calculated value of σ_x at y = 3.85

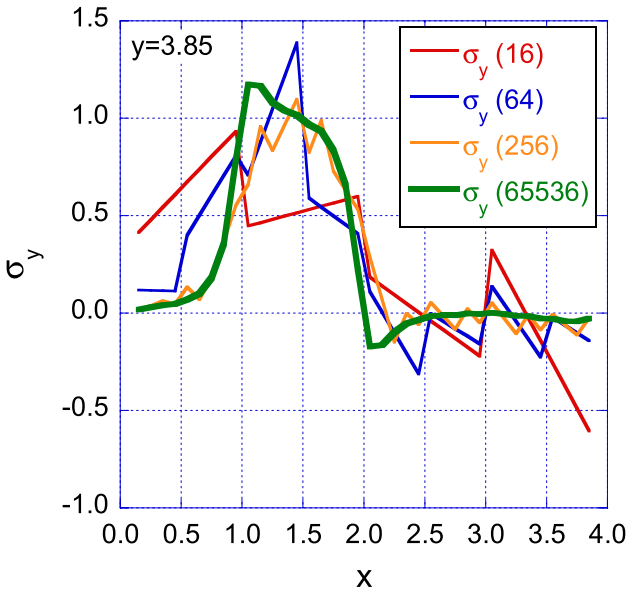

Fig. 31 Calculated value of σ_y at y = 3.85

The stress values at the stress evaluation point group (y = 3.85) given in red in Fig. 7 under the loading condition illustrated in the figure are shown in Fig. 30, where σ_x (16), σ_x(64) and σ_x(65,536) are those with the coarse mesh, the refined mesh and the very fine mesh, respectively. The stresses with the mesh of 256 (16×16) elements (σ_x(256)) are also shown for comparison. In these figures, the vertical axis is the stress, and the horizontal axis the x coordinate of

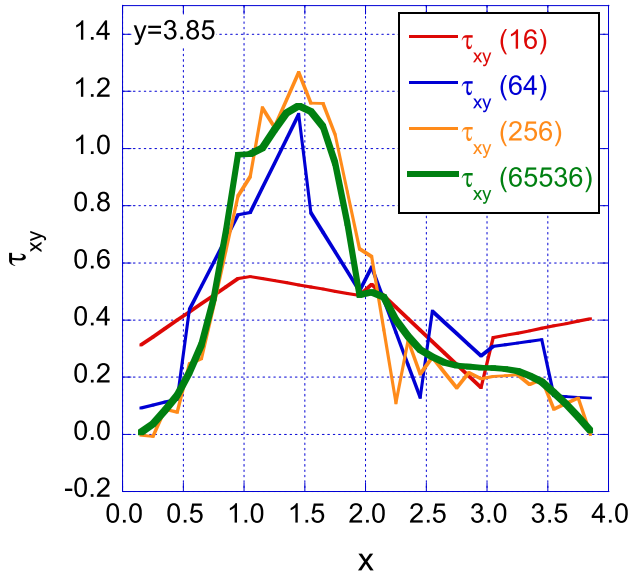

Fig. 32 Calculated value of τ_{xy} at y = 3.85

the stress evaluation point. Shown similarly are Figs. 31 and 32, respectively, for σ_y and τ_{xy}.

4.3.2 Training Patterns for Deep Learning

The training pattern for the deep learning consists of input data to hierarchical neural networks and the corresponding teacher data. The input data in this study are generated from the stress values at five points shown in Fig. 13. Taking a stress evaluation point (P_A) as a point of interest, we employ as the input data the differences between the stress values $(\sigma_x^C, \sigma_y^C, \tau_{xy}^C)$ and $(\sigma_x^M, \sigma_y^M, \tau_{xy}^M)$ at the point and four surrounding points $(P_{A1}, P_{A2}, P_{A3}, P_{A4})$ shown in Fig. 13, resulting in 15 (three stress components × 5 points) pieces of data, and those between $(\sigma_x^M, \sigma_y^M, \tau_{xy}^M)$ at the point and $(\sigma_x^M, \sigma_y^M, \tau_{xy}^M)$ at each of the four surrounding points, resulting in 12 (three stress components × 4 points) pieces of data.

As the corresponding teacher signals, we employ the differences between the stress values $(\sigma_x^M, \sigma_y^M, \tau_{xy}^M)$ and $(\sigma_x^F, \sigma_y^F, \tau_{xy}^F)$ at the point. Table 3 summarizes the 27 input data, and Table 4 the three teacher signals, where ${}^{P_A}\sigma_x^M$ is σ_x^M at the point P_A and so on.

Table 4 Teacher signals for deep learning (Method B)

${}^{P_A}\sigma_x^F - {}^{P_A}\sigma_x^M$	${}^{P_A}\sigma_y^F - {}^{P_A}\sigma_y^M$	${}^{P_A}\tau_{xy}^F - {}^{P_A}\tau_{xy}^M$

All the input data above are generated from the stress values obtained by the analysis with a coarse mesh and its refined mesh, that is, the analysis results obtained by each cycle in adaptive method, while the teacher signals include analysis results with very fine mesh.

As described in Sect. 3.3, 1444 points out of 1600 stress evaluation points excluding the outermost circumference are selected as the point of interest, which results in 1444 training patterns per each analysis. Thus, a total of about 9.36 million (= 6480 × 1444) training patterns are prepared. Note that all the data in the training patterns are normalized to the range of [0, 1].

4.3.3 Deep Learning

Extracted at random from about 9360,000 patterns prepared in the previous section are the training patterns to train the neural network and the test patterns to verify generalization capability. Tested are four different numbers of training patterns: 300,000, 200,000, 100,000 and 50,000, respectively. The number of test patterns is set to 100,000. The hierarchical neural network employed in this study has 27 units as the input layer and 3 units as the output layer, and the number of hidden layers is varied from 1 to 6, the number of units in each hidden layer from 20, 50, or 80 and the number of training epochs is set to 10,000.

Figure 33 shows how the number of units per hidden layer and that of hidden layers influences the estimation error, when using 300,000 training patterns, where the vertical axis is the error (Eq. 3) for 100,000 test patterns, the horizontal axis the number of hidden layers, HU20, HU50 and HU80 mean that the numbers of units per hidden layer are 20, 50 and 80, respectively, and TP300 means that the number of learning patterns is 300,000. It is seen from the figure that the smallest error is obtained when the number of hidden layers is 4 and the number of units per hidden layer is 80.

Table 3 Input data for deep learning (Method B)

${}^{P_A}\sigma_x^M - {}^{P_A}\sigma_x^C$	${}^{P_{A1}}\sigma_x^M - {}^{P_{A1}}\sigma_x^C$	${}^{P_{A2}}\sigma_x^M - {}^{P_{A2}}\sigma_x^C$	${}^{P_{A3}}\sigma_x^M - {}^{P_{A3}}\sigma_x^C$	${}^{P_{A4}}\sigma_x^M - {}^{P_{A4}}\sigma_x^C$
${}^{P_A}\sigma_y^M - {}^{P_A}\sigma_y^C$	${}^{P_{A1}}\sigma_y^M - {}^{P_{A1}}\sigma_y^C$	${}^{P_{A2}}\sigma_y^M - {}^{P_{A2}}\sigma_y^C$	${}^{P_{A3}}\sigma_y^M - {}^{P_{A3}}\sigma_y^C$	${}^{P_{A4}}\sigma_y^M - {}^{P_{A4}}\sigma_y^C$
${}^{P_A}\tau_{xy}^M - {}^{P_A}\tau_{xy}^C$	${}^{P_{A1}}\tau_{xy}^M - {}^{P_{A1}}\tau_{xy}^C$	${}^{P_{A2}}\tau_{xy}^M - {}^{P_{A2}}\tau_{xy}^C$	${}^{P_{A3}}\tau_{xy}^M - {}^{P_{A3}}\tau_{xy}^C$	${}^{P_{A4}}\tau_{xy}^M - {}^{P_{A4}}\tau_{xy}^C$
${}^{P_{A1}}\sigma_x^M - {}^{P_A}\sigma_x^M$	${}^{P_{A2}}\sigma_x^M - {}^{P_A}\sigma_x^M$	${}^{P_{A3}}\sigma_x^M - {}^{P_A}\sigma_x^M$	${}^{P_{A4}}\sigma_x^M - {}^{P_A}\sigma_x^M$	
${}^{P_{A1}}\sigma_y^M - {}^{P_A}\sigma_y^M$	${}^{P_{A2}}\sigma_y^M - {}^{P_A}\sigma_y^M$	${}^{P_{A3}}\sigma_y^M - {}^{P_A}\sigma_y^M$	${}^{P_{A4}}\sigma_y^M - {}^{P_A}\sigma_y^M$	
${}^{P_{A1}}\tau_{xy}^M - {}^{P_A}\tau_{xy}^M$	${}^{P_{A2}}\tau_{xy}^M - {}^{P_A}\tau_{xy}^M$	${}^{P_{A3}}\tau_{xy}^M - {}^{P_A}\tau_{xy}^M$	${}^{P_{A4}}\tau_{xy}^M - {}^{P_A}\tau_{xy}^M$	

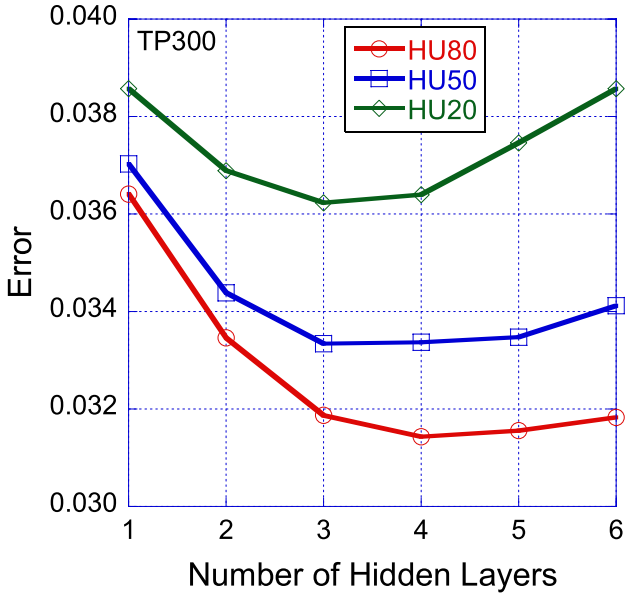

Fig. 33 Errors versus number of hidden layers for various number of units (Method B)

Figure 34 shows how the number of training patterns influences the error (Eq. 3) with the number of units per hidden layer set to 80, where TP300, TP200, TP100 and TP050 mean that the numbers of training patterns are 300,000, 200,000, 100,000 and 50,000, respectively, and the vertical axis is the error for the test pattern. It can be seen from the figure that the error for the test patterns decreases as the number of training patterns increases.

Based on the above results, we conclude that the neural network with 4 hidden layers and 80 units per hidden layer is the best for this sample analysis. When the number of training epochs is extended to 30,000 using the neural network of the best structure, the error for the test patterns is reduced from 0.03143 at 10,000 epochs to 0.03027 at 30,000 epochs.

4.3.4 Accuracy of Proposed Method

Evaluated here is the accuracy of the best hierarchical neural network with the smallest error, trained through 30,000 epochs above. Figure 35 shows the distribution of estimation errors for 100,000 test patterns, where the vertical axis is the number of patterns and the horizontal axis the absolute value of the difference between the stress values estimated by the deep learning and those obtained by the fine mesh of 65,536 elements. When the difference is in the range of [0.00, 0.02], the median of 0.01 is taken as the representative value and so on. All patterns with an absolute value of error of 0.98 or more are considered to have a representative value of 0.99.

For each stress component, the distribution of the absolute value of the difference between the stress value obtained by the very fine mesh of 65,536 elements and that by the refined mesh of 64 elements division is indicated by a dotted line for comparison. It is seen from the figure that the errors of stresses estimated by the deep learning are mostly close to 0.0 and the number of patterns that result in relatively large

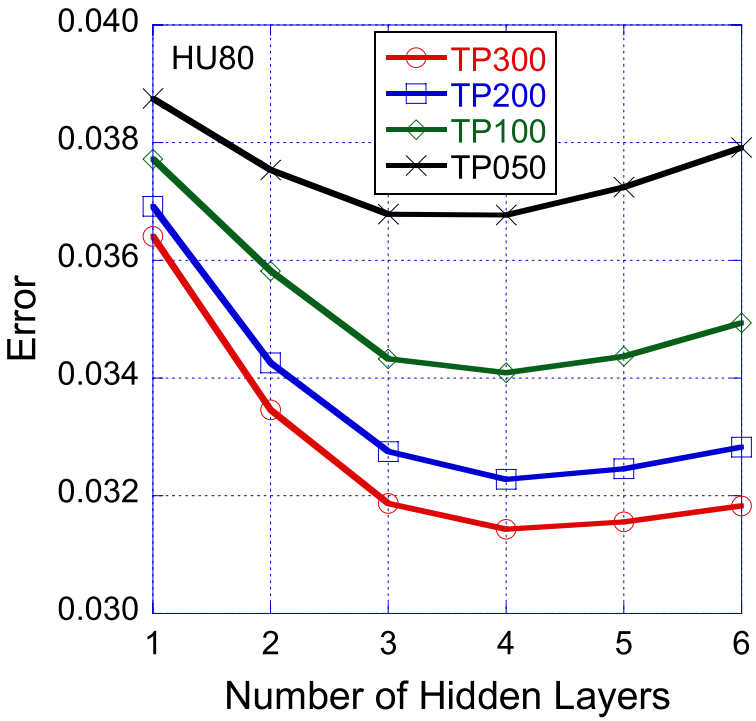

Fig. 34 Errors vs. number of hidden layers for various number of training patterns (Method B)

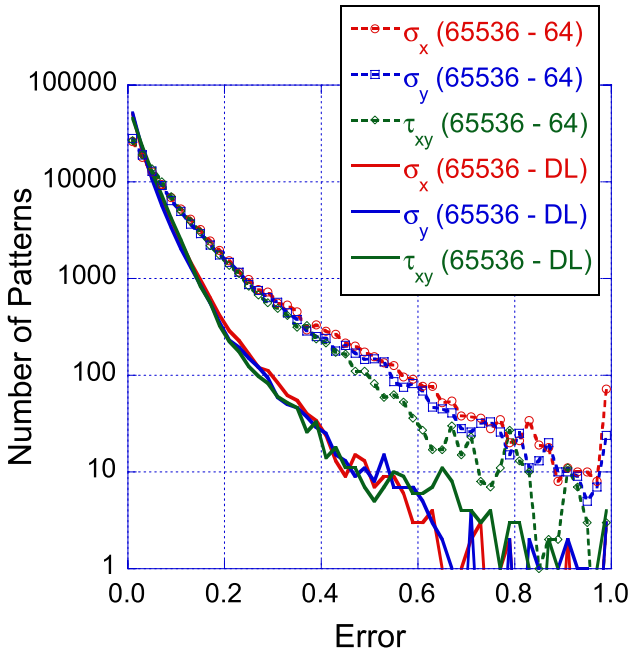

Fig. 35 Number of test patterns versus error (Method B)

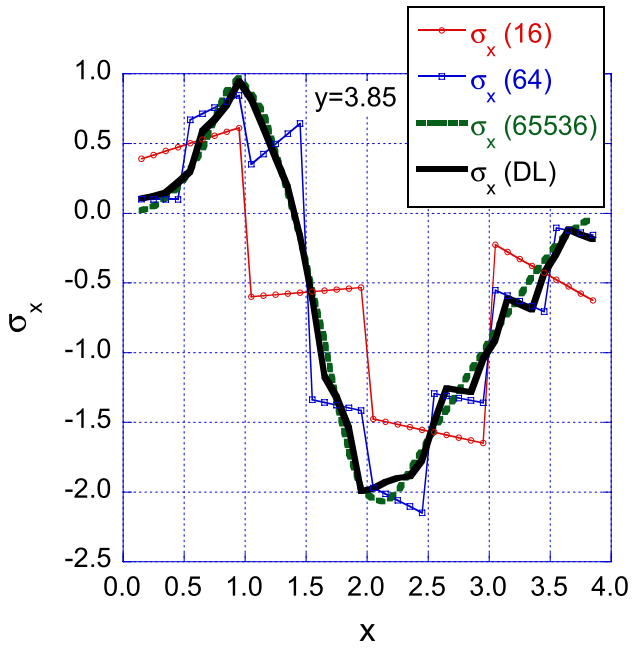

Fig. 36 Estimated value of σ_x at y = 3.85 (Method B)

errors is less than one-tenth compared to that by the refined mesh of 64 elements.

The stresses estimated using the trained neural network are shown for the analysis condition illustrated in Fig. 7 in what follows. For the group of 38 stress evaluation points (y = 3.85) shown in red in Figs. 7, Figs. 36, 37 and 38 show, respectively, the estimation results for σ_x, σ_y, and τ_{xy}. In these

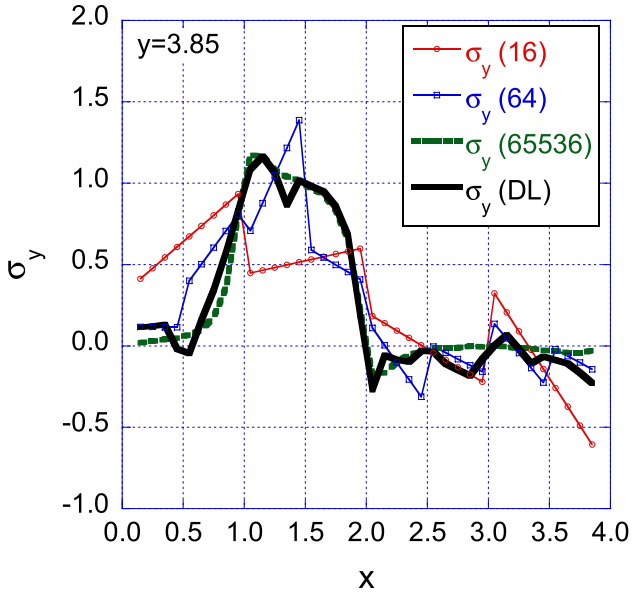

Fig. 37 Estimated value of σ_y at y = 3.85 (Method B)

figures, the stress values obtained by the coarse, refined, and very fine mesh are plotted for comparison. It can be seen from these figures that the stresses estimated by the deep learning coincide well with those by very fine mesh. Note also that the results shown in Figs. 36, 37 and 38 are those not included in training patterns, indicating a high versatility of the proposed method.

5 Conclusions

In this paper, we propose two kinds of high-performance finite element methods using the error information with neural networks.

The neural network is applied to improve the accuracy of the finite element solution using the error information obtained by a posterior error estimation in the Method A, where the neural network and the finite element analysis with the coarse mesh can output accurate stresses using only the information on the local stress state and its error in the vicinity of the target point.

In the Method B, the neural network is employed to improve the adaptive finite element method, in which the neural network is trained to predict much more accurate stresses that otherwise would require time-consuming analysis using the error information obtained from the analysis results varying meshes.

One of the advantages of both methods is their versatility that they only use information of local stress state as input to the neural network. The other advantage of the proposed methods is their speed that they can estimate highly accurate stresses of the specified region of an object by several orders

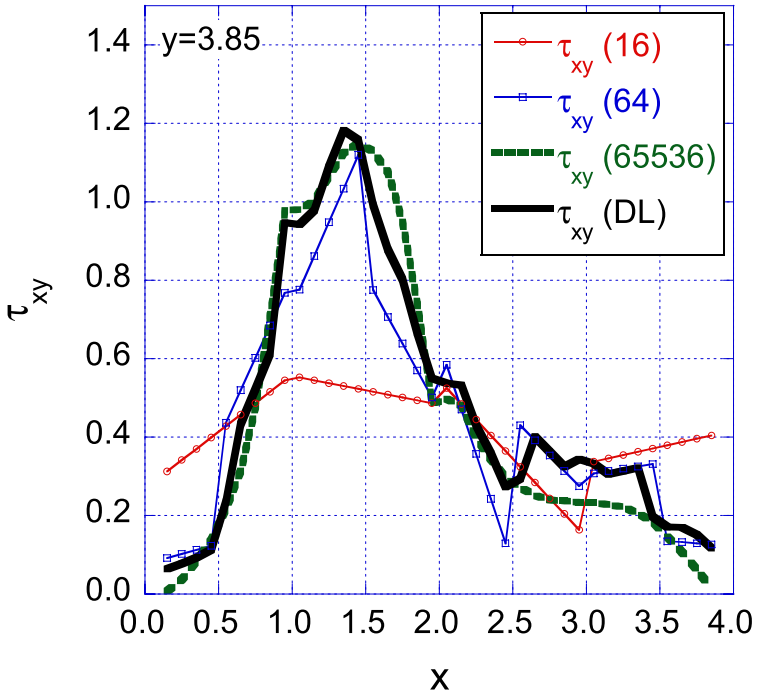

Fig. 38 Estimated value of τ_{xy} at y = 3.85 (Method B)

of magnitude faster than the conventional finite element method, which is very attractive especially for the structural optimization where a lot of stress analyses are required varying shapes and topologies of the structure.

Though the proposed methods are considered very promising, they have some issues to be solved. One of them will be the collection of training data which requires very heavy computational power. These issues remain as future works.

Funding The authors received no specific funding for this work.

Compliance with Ethical Standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Ainsworth M, Oden JT (2000) A Posteriori error estimation in finite element Analysis. Wiley, New York
2. Akiba H, Ohyama T, Shibata Y, Yuyama K, Katai Y, Takeuchi R, Hoshino T, Yoshimura S, Noguchi H, Gupta M, Gunnels JA, Austel V, Sabharwal Y, Garg R, Kato S, Kawakami T, Todokoro S, Ikeda J (2006) Large scale drop impact analysis of mobile phone using ADVC on Blue Gene/L. In: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, Tampa, Florida https://doi.org/10.1145/1188455.1188503
3. Al-Haik MS, Garmestani H, Navon IM (2003) Truncated-Newton training algorithm for neurocomputational viscoplastic model. Comput Methods Appl Mech Eng 192:2249–2267
4. Arai K, Yodo K, Okada H, Yamada T, Kawai H, Yoshimura S (2015) Ultra-large scale fracture mechanics analysis using a parallel finite element method with submodel technique. Finite Elem Anal Des 105:44–55
5. Babuska I, Rheinboldt WC (1978) Error estimates for adaptive finite element computations. SIAM J Numer Anal 15:736–754
6. Babuska I, Vogelius M (1984) Feedback and adaptive finite element solution of one-dimensional boundary value problems. Numer Math 44:75–102
7. Bathe KJ (1996) Finite element procedures. Prentice-Hall, Upper Saddle River
8. Bengio Y, Lamblin P, Popovici D, Larochelle H (2006) Greedy layer-wise training of deep networks. In: Proceedings of NIPS, 2006
9. Benson DJ, Hallquist JO (1990) A single surface contact algorithm for the post-buckling analysis of shell structures. Comput Methods Appl Mech Eng 78:141–163
10. Bhatnagar S, Afshar Y, Pan S, Duraisamy K, Kaushik S (2019) Prediction of aerodynamic flow fields using convolutional neural networks. Comput Mech 64:525–545
11. Binev P, Dahmen W, DeVore R (2004) Adaptive finite element methods with convergence rates. Numer Math 97(2):219–268
12. Bishop CM (2006) Pattern recognition and machine learning. Springer, New York
13. Biswas R, Devine KD, Flaherty JE (1994) Parallel, adaptive finite element methods for conservation laws. Appl Numer Math 14:255–283
14. Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm intelligence: from natural to artificial systems. Oxford University Press, Oxford
15. Buffa G, Patrinostro G, Fratini L (2014) Using a neural network for qualitative and quantitative predictions of weld integrity in solid bonding dominated processes. Comput Struct 135:1–9
16. Bugeda G, Ródenas JJ, Albelda J, Oñate E (2009) Control of the finite element discretization error during the convergence of structural shape optimization algorithms. Int J Simul Multidiscipl Des Optim 3(3):363–369

17. Bugeda G, Ródenas JJ, Oñate E (2008) An integration of a low cost adaptive remeshing strategy in the solution of structural shape optimization problems using evolutionary methods. Comput Struct 86(13–14):1563–1578
18. Cang R, Li H, Yao H, Jiao Y, Rena Y (2018) Improving direct physical properties prediction of heterogeneous materials from imaging data via convolutional neural network and a morphology-aware generative model. Comput Mater Sci 150:212–221
19. Cao X, Sugiyama Y, Mitsui Y (1998) Application of artificial neural networks to load identification. Comput Struct 69:63–78
20. Carey GF (1997) Computational grids: generation, adaptation, and solution strategies. Taylor & Francis, Washington, D.C.
21. Carstensen C, Feischl M, Page M, Praetorius D (2014) Axioms of adaptivity. Comput Math Appl 67(6):1195–1253
22. Cascon JM, Kreuzer C, Nochetto RH, Siebert KG (2008) Quasi-optimal convergence rate for an adaptive finite element method. SIAM J Numer Anal 46(5):2524–2550
23. Cecka C, Lew AJ, Darve E (2011) Assembly of finite element methods on graphics processors. Int J Numer Methods Eng 85:640–669
24. Chakraverty S, Singh VP, Sharma RK (2006) Regression based weight generation algorithm in neural network for estimation of frequencies of vibrating plates. Comput Methods Appl Mech Eng 195:4194–4202
25. Cheng J, Li QS (2008) Reliability analysis of structures using artificial neural network based genetic algorithms. Comput Methods Appl Mech Eng 197:3742–3750
26. Cheng J, Li QS (2009) A hybrid artificial neural network method with uniform design for structural optimization. Comput Mech 44:61–71
27. Cheng Z, Wang H (2018) How to control the crack to propagate along the specified path feasibly. Comput Methods Appl Mech Eng 336:554–577
28. Cho JR, Shin SW, Yoo WS (2005) Crown shape optimization for enhancing tire wear performance by ANN. Comput Struct 83:920–933
29. Chowdhury A, Kautz E, Yener B, Lewis D (2016) Image driven machine learning methods for microstructure recognition. Comput Mater Sci 123:176–187
30. Dadvand P, Rossi R, Oñate E (2010) An object oriented environment for developing finite element codes for multi-disciplinary applications. Arch Comput Methods Eng 17(3):253–297
31. Daoheng S, Qiao H, Hao X (2000) A neurocomputing model for the elastoplasticity. Comput Methods Appl Mech Eng 182:177–186
32. Dörfler W (1996) A convergent adaptive algorithm for Poisson's equation. SIAM J Numer Anal 33:1106–1124
33. Finol D, Lu Y, Mahadevan V, Srivastava A (2019) Deep convolutional neural networks for eigenvalue problems in mechanics. Int J Numer Methods Eng 118:258–275
34. Flood I, Muszynski L, Nandy S (2001) Rapid analysis of externally reinforced concrete beams using neural networks. Comput Struct 79:1553–1559
35. Fratini L, Buffa G, Palmeri D (2009) Using a neural network for predicting the average grain size in friction stir welding processes. Comput Struct 87:1166–1174
36. Freitag S, Cao BT, Ninic J, Meschke G (2018) Recurrent neural networks and proper orthogonal decomposition with interval data for real-time predictions of mechanised tunnelling processes. Comput Struct 207:258–273
37. Freitag S, Graf W, Kaliske M (2013) A material description based on recurrent neural networks for fuzzy data and its application within the finite element method. Comput Struct 124:29–37
38. Funahashi K (1989) On the approximate realization of continuous mappings by neural networks. Neural Netw 2:183–192
39. Furukawa T, Yagawa G (1998) Implicit constitutive modelling for viscoplasticity using neural networks. Int J Numer Methods Eng 43:195–219
40. Garatani K, Nakajima K, Okuda H, Yagawa G (2001) Three-dimensional elasto-static analysis of 100 million degrees of freedom. Adv Eng Softw 32:511–518
41. Garijo N, Martinez J, Garcia-Aznar JM, Perez MA (2014) Computational evaluation of different numerical tools for the prediction of proximal femur loads from bone morphology. Comput Methods Appl Mech Eng 268:437–450
42. Gawin D, Lefik M, Schrefler BA (2001) ANN approach to sorption hysteresis within a coupled hygro-thermo-mechanical FE analysis. Int J Numer Methods Eng 50:299–323
43. Ghaboussi J, Pecknold DA, Zhang M, Haj-Ali R (1998) Autoprogressive training of neural network constitutive models. Int J Numer Methods Eng 42:105–126
44. Ghosh DK, Basu PK (1994) Parallel adaptive finite element analysis of large scale structures. Comput Sys Eng 5:325–335
45. Goldberg DE (1989) Genetic algorithms in search. Addison-Wesley, Optimization & Machine Learning
46. Gomes WJS, Beck AT (2013) Global structural optimization considering expected consequences of failure and using ANN surrogates. Comput Struct 126:56–68
47. Gonzalez MP, Zapico JL (2008) Seismic damage identification in buildings using neural networks and modal data. Comput Struct 86:416–426
48. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge
49. Grätsch T, Bathe KJ (2005) A posteriori error estimation techniques in practical finite element analysis. Comput Struct 83:235–265
50. Hallquist JO, Goudreau GL, Benson DJ (1985) Sliding interfaces with contact-impact in large-scale Lagrangian computations. Comput Methods Appl Mech Eng 51:107–137
51. Hambli R (2009) Statistical damage analysis of extrusion processes using finite element method and neural networks simulation. Finite Elem Anal Des 45:640–649
52. Hambli R (2011) Numerical procedure for multiscale bone adaptation prediction based on neural networks and finite element simulation. Finite Elem Anal Des 47:835–842
53. Hambli R, Chamekh A, Salah HBH (2006) Real-time deformation of structure using finite element and neural networks in virtual reality applications. Finite Elem Anal Des 42:985–991
54. Hambli R, Guerin F (2003) Application of neural network for optimum clearance prediction in sheet metal blanking processes. Finite Elem Anal Des 39:1039–1052
55. Hasançebi O, Dumlupınar T (2013) Detailed load rating analyses of bridge populations using nonlinear finite element models and artificial neural networks. Comput Struct 128:48–63
56. Hasançebi O, Dumlupınar T (2013) Linear and nonlinear model updating of reinforced concrete T-beam bridges using artificial neural networks. Comput Struct 119:1–11
57. Hashash YMA, Jung S, Ghaboussi J (2004) Numerical implementation of a network based material model in finite element analysis. Int J Numer Methods Eng 59:989–1005
58. Hattori G, Serpa AL (2015) Contact stiffness estimation in ANSYS using simplified models and artificial neural networks. Finite Elem Anal Des 97:43–53
59. Heykin S (1999) Neural networks: a comprehensive foundation, 2nd edn. Prentice Hall, Upper Saddle River
60. Hinton GE, Osindero S, Teh Y (2006) A fast learning algorithm for deep belief nets. Neural Comput 18:1527–1544
61. Hirschen K, Schafer M (2006) Bayesian regularization neural networks for optimizing fluid flow processes. Comput Methods Appl Mech Eng 195:481–500

62. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2:359–366
63. Huber N, Tsakmakis C (2001) A neural network tool for identifying the material parameters of a finite deformation viscoplasticity model with static recovery. Comput Methods Appl Mech Eng 191:353–384
64. Hughes TJR (2000) The finite element method: linear static and dynamic finite element analysis. Dover, New York
65. Hurtado JE, Alvarez DA (2001) Neural-network-based reliability analysis: a comparative study. Comput Methods Appl Mech Eng 191:113–132
66. Johnson AA, Tezduyar TE (1999) Advanced mesh generation and update methods for 3d flow simulations. Comput Mech 23(2):130–143
67. Jenkins WM (2006) Neural network weight training by mutation. Comput Struct 84:2107–2112
68. Jung S, Ghaboussi J (2006) Characterizing rate-dependent material behaviors in self-learning simulation. Comput Methods Appl Mech Eng 196:608–619
69. Jung S, Ghaboussi J (2006) Neural network constitutive model for rate-dependent materials. Comput Struct 84:955–963
70. Kallassy A (2003) A new neural network for response estimation. Comput Struct 81:2417–2429
71. Kao CY, Hung SL (2003) Detection of structural damage via free vibration responses generated by approximating artificial neural networks. Comput Struct 81:2631–2644
72. Kaveh A, Bahreininejad A, Mostafaei H (1999) A hybrid graph-neural method for domain decomposition. Comput Struct 70:667–674
73. Kaveh A, Rahimi Bondarabady HA (2004) Wavefront reduction using graphs, neural networks and genetic algorithm. Int J Numer Methods Eng 60:1803–1815
74. Kaveh A, Servati H (2001) Design of double layer grids using backpropagation neural networks. Comput Struct 79:1561–1568
75. Kawai H, Ogino M, Shioya R, Yoshimura S (2011) Large scale elasto-plastic analysis using domain decomposition method optimized for multi-core CPU architecture. Key Eng Mater 462–463:605–610
76. Kikuchi M (1989) Application of the symbolic mathematics system to the finite element program. Comput Mech 5:41–47
77. Kim JH, Kim YH (2001) A predictor-corrector method for structural nonlinear analysis. Comput Methods Appl Mech Eng 191:959–974
78. Klaas O, Niekamp R, Stein E (1995) Parallel adaptive finite element computations with hierarchical preconditioning. Comput Mech 16(1):45–52
79. Klos M, Waszczyszyn Z (2011) Modal analysis and modified cascade neural networks in identification of geometrical parameters of circular arches. Comput Struct 89:581–589
80. Kondo R, Yamakawa S, Masuoka Y, Tajima S, Asahi R (2017) Microstructure recognition using convolutional neural networks for prediction of ionic conductivity in ceramics. Acta Mater 141:29–38
81. Kouhi M, Dehghan Manshadi M, Oñate E (2014) Geometry optimization of the diffuser for the supersonic wind tunnel using genetic algorithm and adaptive mesh refinement technique. Aerosp Sci Technol 36:64–74
82. Kouhi M, Lee DS, Bugeda G, Oñate E (2013) Multi-objective aerodynamic shape optimization using MOGA coupled to advanced adaptive mesh refinement. Comput Fluids 88:298–312
83. Koza JR (1992) Genetic programming. MIT Press, Cambridge
84. Koza JR (1994) Genetic programming II. MIT Press, Cambridge
85. Kubo S (1988) Inverse problems related to the mechanics and fracture of solids and structures. JSME Int J 31(2):157–166
86. Lagaros ND, Charmpis DC, Papadrakakis M (2005) An adaptive neural network strategy for improving the computational performance of evolutionary structural optimization. Comput Methods Appl Mech Eng 194:3374–3393
87. Lagaros ND, Garavelas AT, Papadrakakis M (2008) Innovative seismic design optimization with reliability constraints. Comput Methods Appl Mech Eng 198:28–41
88. Lanzi L, Bisagni C, Ricci S (2004) Neural network systems to reproduce crash behavior of structural components. Comput Struct 82:93–108
89. Le BA, Yvonnet J, He QC (2015) Computational homogenization of nonlinear elastic materials using neural networks. Int J Numer Methods Eng 104:1061–1084
90. Lee J, Jeong H, Choi DH, Volovoi V, Marvis D (2007) An enhancement of consistent feasibility in BPN based approximate optimization. Comput Methods Appl Mech Eng 196:2147–2160
91. Lee SC, Park SK, Lee BH (2001) Development of the approximate analytical model for the stub-girder system using neural network. Comput Struct 79:1013–1025
92. Lefik M, Boso DP, Schrefler BA (2009) Artificial neural networks in numerical modelling of composites. Comput Methods Appl Mech Eng 198:1785–1804
93. Lefik M, Schrefler BA (2002) Artificial neural network for parameter identifications for an elasto-plastic model of superconducting cable under cyclic loading. Comput Struct 80:1699–1713
94. Lefik M, Schrefler BA (2003) Artificial neural network as an incremental non-linear constitutive model for a finite element code. Comput Methods Appl Mech Eng 192:3265–3283
95. Li S (2000) Global flexibility simulation and element stiffness simulation in finite element analysis with neural network. Comput Methods Appl Mech Eng 186:101–108
96. Li X, Liu Z, Cui S, Luo C, Li C, Zhuang Z (2019) Predicting the effective mechanical property of heterogeneous materials by image based modeling and deep learning. Comput Methods Appl Mech Eng 347:735–753
97. Li ZX, Yang XM (2008) Damage identification for beams using ANN based on statistical property of structural responses. Comput Struct 86:64–71
98. Liang YC, Feng DP, Liu GR, Yang XW, Han X (2003) Neural identification of rock parameters using fuzzy adaptive learning parameters. Comput Struct 81:2373–2382
99. Lin CY, Lin SH (2005) Artificial neural network based hole image interpretation techniques for integrated topology and shape optimization. Comput Methods Appl Mech Eng 194:3817–3837
100. Lin JC (2003) Using FEM and neural network prediction on hydrodynamic deep drawing of T-piece maximum length. Finite Elem Anal Design 39:445–456
101. Liu SW, Huang JH, Sung JC, Lee CC (2002) Detection of cracks using neural networks and computational mechanics. Comput Methods Appl Mech Eng 191:2831–2845
102. Lopez R, Balsa-Canto E, Oñate E (2008) Neural networks for variational problems in engineering. Int J Numer Methods Eng 75:1341–1360
103. López R, Oñate E (2008) An extended class of multilayer perceptron. Neurocomputing 71(13–15):2538–2543
104. Majorana C, Odorizzi S, Vitaliani R (1982) Shortened quadrature rules for finite elements. Adv Eng Softw 4:52–57
105. Man H, Furukawa T (2011) Neural network constitutive modelling for non-linear characterization of anisotropic materials. Int J Numer Methods Eng 85:939–957
106. Markall GR, Slemmer A, Ham DA, Kelly PHJ, Cantwell CD, Sherwin SJ (2013) Finite element assembly strategies on multicore andmany-core architectures. Int J Numer Methods Eng 71:80–97

107. Melenk JM, Gerdes K, Schwab C (2001) Fully discrete hp-finite elements: fast quadrature. Compt Methods Appl Mech Eng 190:4339–4364
108. Michalewicz Z (1992) Genetic Algorithms + Data Structures = Evolution Programs. Springer, New York
109. Mitchell WF (1989) A comparison of adaptive refinement techniques for elliptic problems. ACM Trans Math Softw 15(4):326–347
110. Morin P, Nochetto RH, Siebert KG (2002) Convergence of adaptive finite element methods. SIAM Rev 44(4):631–658
111. Murotani K, Yagawa G, Choi JB (2013) Adaptive finite elements using hierarchical mesh and its application to crack propagation analysis. Comput Methods Appl Mech Eng 253:1–14
112. Murphy KP (2012) Machine learning: a probabilistic perspective. MIT Press, Cambridge
113. Nikolaidis E, Long L, Ling Q (2000) Neural networks and response surface polynomials for design of vehicle joints. Comput Struct 75:593–607
114. Nikolaidis E, Zhu M (1996) Design of automotive joints: using neural networks and optimization to translate performance requirements to physical design parameters. Comput Struct 60(6):989–1001
115. Oeser M, Freitag S (2009) Modeling of materials with fading memory using neural networks. Int J Numer Methods Eng 78:843–862
116. Oishi A, Yagawa G (2017) Computational mechanics enhanced by deep learning. Comput Methods Appl Mech Eng 327:327–351
117. Oishi A, Yagawa G (2020) A surface-to-surface contact search method enhanced by deep learning. Comput Mech 65:1125–1147
118. Oishi A, Yamada K, Yoshimura S, Yagawa G (1995) Quantitative nondestructive evaluation with ultrasonic method using neural networks and computational mechanics. Comput Mech 15:521–533
119. Oishi A, Yamada K, Yoshimura S, Yagawa G, Nagai S, Matsuda Y (2001) Neural network-based inverse analysis for defect identification with laser ultrasonics. Res Nondestruct Eval 13(2):79–95
120. Oishi A, Yoshimura S (2007) A new local contact search method using a multi-layer neural network. Comput Model Eng Sci 21(2):93–103
121. Oishi A, Yoshimura S (2008) Finite element analyses of dynamic problems using graphic hardware. Comput Model Eng Sci 25(2):115–131
122. Okuda H, Yagawa G (1997) Multi-color neural network with feedback mechanism for parallel finite element fluid analysis. In: Papadrakakis M (ed) Parallel solution methods in computational mechanics. Wiley, New York, pp 431–457
123. Okuda H, Yoshimura S, Yagawa G, Matsuda A (1998) Neural network-based parameter estimation for non-linear finite element analyses. Eng Comput 15(1):103–138
124. Oñate E, Arteaga J, García Espinosa J, Flores R (2006) Error estimation and mesh adaptivity in incompressible viscous flows using a residual power approach. Comput Methods Appl Mech Eng 195(4–6):339–362
125. Oñate E, Bugeda G (1993) A study of mesh optimality criteria in adaptive finite element analysis. Eng Comput 10(4):307–321
126. Pandey PC, Barai SV (1995) Multilayer perceptron in damage detection of bridge structures. Comput Struct 54(4):597–608
127. Papadopoulos V, Soimiris G, Giovanis DG, Papadrakakis M (2018) A neural network-based surrogate model for carbon nanotubes with geometric nonlinearities. Comput Methods Appl Mech Eng 328:411–430
128. Papadrakakis M, Lagaros ND (2002) Reliability-based structural optimization using neural networks and Monte Carlo simulation. Comput Methods Appl Mech Eng 191:3491–3507
129. Papadrakakis M, Stravroulakis G, Karatarakis A (2011) A new era in scientific computing: domain decomposition methods in hybrid CPU-GPU architectures. Comput Methods Appl Mech Eng 200:1490–1508
130. Parpinelli RS, Teodoro FR, Lopes HS (2012) A comparison of swarm intelligence algorithms for structural engineering optimization. Int J Numer Methods Eng 91:666–684
131. Patel D, Tibrewala R, Vega A, Dong L, Hugenberg N, Oberai AA (2019) Circumventing the solution of inverse problems in mechanics through deep learning: application to elasticity imaging. Comput Methods Appl Mech Eng 353:448–466
132. Pei JS, Wright JP, Smyth AW (2005) Mapping polynomial fitting into feedforward neural networks for modeling nonlinear dynamic systems and beyond. Comput Methods Appl Mech Eng 194:4481–4505
133. Protopapadakis E, Schauer M, Pierri E, Doulamis AD, Stavroulakis GE, Böhrnsen JU, Langer S (2016) A genetically optimized neural classifier applied to numerical pile integrity tests considering concrete piles. Comput Struct 162:68–79
134. Rafiq MY, Bugmann G, Easterbrook DJ (2001) Neural network design for engineering applications. Comput Struct 79:1541–1552
135. Raissi M, Perdikaris P, Karniadakis GE (2019) Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 378:686–707
136. Ramu SA, Johnson VT (1995) Damage assessment of composite structures—a fuzzy logic integrated neural network approach. Comput Struct 57(3):491–502
137. Rank E, Zienkiewicz OC (1987) A simple error estimator for the finite element method. Commun Appl Numer Methods 3:243–249
138. Rao HS, Ghorpade VG, Mukherjee A (2006) A genetic algorithm based back propagation network for simulation of stress–strain response of ceramic-matrix-composites. Comput Struct 84:330–339
139. Rhim J, Lee SW (1995) A neural network approach for damage detection and identification of structures. Comput Mech 16:437–443
140. Rivara MC (1984) Design and data structure for fully adaptive, multigrid finite element software. ACM Trans Math Softw 10:242–264
141. Rivara MC (1984) Mesh refinement processes based on the generalized bisection of simplices. SIAM J Numer Anal 21:604–613
142. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323:533–536
143. Salazar F, Morán R, Rossi R, Oñate E (2013) Analysis of the discharge capacity of radial-gated spillways using CFD and ANN—Oliana Dam case study. J Hydraul Res 51(3):244–252
144. Salazar F, Moran R, Toledo MA, Oñate E (2017) Data-based models for the prediction of dam behaviour: a review and some methodological considerations. Arch Comput Methods Eng 24(1):1–21
145. Salazar F, Toledo MA, Morán R, Oñate E (2015) An empirical comparison of machine learning techniques for dam behaviour modelling. Struct Saf 56:9–17
146. Slonski M (2010) A comparison of model selection methods for compressive strength prediction of high-performance concrete using neural networks. Comput Struct 88:1248–1253
147. Stevenson R (2007) Optimality of a standard adaptive finite element method. Found Comput Math 7(2):245–269
148. Suk JW, Kim JH, Kim YH (2003) A predictor algorithm for fast geometrically-nonlinear dynamic analysis. Comput Methods Appl Mech Eng 192:2521–2538
149. Sumelka W, Lodygowski T (2013) Reduction of the number of material parameters by ANN approximation. Comput Mech 52:287–300

150. Tang YC, Zhou XH, Chen J (2008) Preform tool shape optimization and redesign based on neural network response surface methodology. Finite Elem Anal Des 44:462–471
151. Tezduyar TE, Aliabadi S, Behr M (1998) Enhanced-discretization interface-capturing technique (edict) for computation of unsteady flows with interfaces. Comput Methods Appl Mech Eng 155:235–248
152. Theocaris PS, Panagiotopoulos PD (1995) Generalised hardening plasticity approximated via anisotropic elasticity: a neural network approach. Comput Methods Appl Mech Eng 125:123–139
153. Toparli M, Sahin S, Ozkaya E, Sasaki S (2002) Residual thermal stress analysis in cylindrical steel bars using finite element method and artificial neural networks. Comput Struct 80:1763–1770
154. Unger JF, Könke C (2008) Coupling of scales in a multiscale simulation using neural networks. Comput Struct 86:1994–2003
155. Unger JF, Könke C (2009) Neural networks as material models within a multiscale approach. Comput Struct 87:1177–1186
156. Verfürth R (1996) A review of a posteriori error estimation and adaptive mesh refinement techniques. Wiley, New York
157. Verfürth R (2013) A posteriori error estimation techniques for finite element methods. Oxford University Press, Oxford
158. Wang K, Sun WC (2018) A multiscale multi-permeability poroplasticity model linked by recursive homogenizations and deep learning. Comput Methods Appl Mech Eng 334:337–380
159. Yagawa G, Aoki O (1995) A neural network-based finite element method on parallel processors. Contemporary research in engineering science. Springer, New York, pp 637–653
160. Yagawa G, Ichimiya M, Ando Y (1978) Analysis method for stress intensity factors based on the discretization error in the finite element method. Trans JSME 44(379):743–755 **(in Japanese)**
161. Yagawa G, Okuda H (1996) Neural networks in computational mechanics. Arch Comput Methods Eng 3(4):435–512
162. Yagawa G, Okuda H (1996) Finite element solutions with feedback network mechanism through direct minimization of energy functionals. Int J Numer Methods Eng 39:867–883
163. Yagawa G, Ye GGW, Yoshimura S (1990) A numerical integration scheme for finite element method based on symbolic manipulation. Int J Numer Methods Eng 29:1539–1549
164. Yagawa G, Yoshioka A, Yoshimura S, Soneda N (1993) A parallel finite element method with a supercomputer network. Comput Struct 47(3):407–418
165. Yang Z, Yabansu YC, Al-Bahrania R, Liaoa WK, Choudhary AN, Kalidindi SR, Agrawal A (2018) Deep learning approaches for mining structure-property linkages in high contrast composites from simulation datasets. Comput Mater Sci 151:278–287
166. Yilbas Z, Hashmi MSJ (1998) Simulation of weight pruning process in backpropagation neural network for pattern classification: a self-running threshold approach. Comput Methods Appl Mech Eng 166:233–246
167. Yoshimura S, Matsuda A, Yagawa G (1996) New regularization by transformation for neural network based inverse analyses and its application to structure identification. Int J Numer Methods Eng 39:3953–3968
168. Yun CB, Bahng EY (2000) Substructural identification using neural networks. Comput Struct 77:41–52
169. Yun GJ, Ghaboussi J, Elnashai AS (2008) Self-learning simulation method for inverse nonlinear modeling of cyclic behavior of connections. Comput Methods Appl Mech Eng 197:2836–2857
170. Zacharias J, Hartmann C, Delgado A (2004) Damage detection on crates of beverages by artificial neural networks trained with finite-element data. Comput Methods Appl Mech Eng 193:561–574
171. Zhang L, Subbarayan G (2002) An evaluation of back-propagation neural networks for the optimal design of structural systems: part I. Training procedures. Comput Methods Appl Mech Eng 191:2873–2886
172. Zhang L, Subbarayan G (2002) An evaluation of back-propagation neural networks for the optimal design of structural systems: part II. Numerical evaluation. Comput Methods Appl Mech Eng 191:2887–2904
173. Ziemianski L (2003) Hybrid neural network/finite element modelling of wave propagation in infinite domains. Comput Struct 81:1099–1109
174. Zienkiewicz OC, Taylor RL, Zhu JZ (2005) The finite element method: its basis and fundamentals. Elsevier, New York
175. Zienkiewicz OC, Zhu JZ (1987) A simple error estimator and adaptive procedure for practical engineering analysis. Int J Numer Methods Eng 24:337–357
176. Zopf C, Kaliske M (2017) Numerical characterization of uncured elastomers by a neural network based approach. Comput Struct 182:504–525