Jurica Sorić
Matej Stanić
Tomislav Lesičar ✉

https://doi.org/10.21278/TOF.472053023
ISSN 1333-1124
eISSN 1849-1391

ON NEURAL NETWORK APPLICATION IN SOLID MECHANICS

Summary

A review of the machine learning methods employing the neural network algorithm is presented. Most commonly used neural networks, such as the feedforward neural network including deep learning, the convolutional neural network, the recurrent neural network and the physics-informed neural network, are discussed. A special emphasis is placed on their applications in engineering fields, particularly in solid mechanics. Network architectures comprising layers and neurons as well as different learning processes are highlighted. The feedforward neural network and the recurrent neural network are described in more details. To reduce the undesired vanishing gradient effect within the recurrent neural network architecture, the long short-term memory network is presented. Numerical efficiency and accuracy of both the feedforward and the long short-term memory recurrent network are demonstrated by numerical examples, where the neural network solutions are compared to the results obtained using the standard finite element approaches.

Key words: *machine learning, neural networks, feedforward neural network, recurrent neural network, solid mechanics*

1. Introduction

1.1 Artificial intelligence and machine learning

Artificial intelligence (AI) is based on algorithms by means of which machines perform cognitive tasks like a human brain. An AI approach includes three main steps: storing knowledge, applying knowledge to support problem-solving and acquiring new knowledge through experience, as described in more details in [1]. This approach deals with numerous algorithms to interpret and attain knowledge from given data. In recent years, special attention has been directed to the application of AI methods to solve various engineering problems. The machine learning has evolved as a subfield of AI.

The machine learning (ML) has attracted great attention with the progressive development of computing technology which enables one to address a variety of complex problems by means of collecting and processing a large amount of data. However, these complex problems cannot be solved using conventional procedures, or their solving results in high computational costs. The ML applications are based on statistical algorithms to improve their performance by training [2, 3]. Using enough data and appropriate algorithms, by means

of ML it is possible to determine all known physical laws as well as those that are currently unknown. In this manner, the ML algorithms may be considered as a data driven approach.

Successful applications of ML techniques can be found in various fields, such as neuroscience [4], clinical medicine [5], biotechnology including the design of new drugs [6], robotics applications [7], autonomous driving [8, 9], face recognition [10, 11], speech recognition [12, 13] as well as other fields, such as climate change, earthquake detection, economics, different games, various data handling, etc. Furthermore, a great challenge is the ML application to solve problems in engineering and physical disciplines, such as mechanics [14, 15], materials science [16, 17, 18, 19], topology optimization [16, 20] and other engineering fields.

The machine learning comprises different models which learn from given input data. Depending on the type and amount of available data, an ML model may be supervised, semi-supervised or unsupervised [3]. The ML is supervised if the training data consists of sets of input and associated output values. The goal of the ML algorithm is to derive a function which predicts the output values from a given set of input data. If the available data are only input values, the ML is unsupervised. The ML is semi-supervised if a large amount of input data but only a limited amount of corresponding output data is given. The supervised learning is mostly used, especially in engineering and physical sciences, because it is much more efficient and powerful in comparison to other ML approaches. A great challenge in the ML technique is how to efficiently reduce the training error. The underfitting and overfitting are distinguished. When the model is not able to obtain a sufficiently low error, the underfitting happens. The overfitting occurs when the difference between the training error and the test error is too large [21]. Depending on the type of the data set collected and the problem posed, a countless number of ML algorithms is available. Neural networks are most commonly used.

1.2 Neural networks

Neural networks (NNs) approximately simulate the operation of human brain by means of artificial neurons, which are also called processing units [1]. The neurons are assembled into network architectures which are mostly created in form of layers. There are input, output, and hidden layers. From the source neurons of the input layer, the signal flows over the hidden layers to the layer of output neurons, reproducing an overall response of the network. The layers are called hidden because the training data do not show the output for each of these layers. The function of the hidden layers is to intervene between the external input and the output of the network. Each neuron receives input signals from other neurons of a preceding layer during the learning process. The neuron outputs are computed by appropriate activation functions which are also called transfer functions. There are network parameters such as the connection weight between neurons and the bias which should be adjusted in training procedures to reproduce the training data as accurately as possible [1, 22, 23]. Depending on the network architecture, there are many kinds of neural network algorithms. The most fundamental type of NNs is the feedforward neural network.

1.2.1 Feedforward neural network

In the feedforward neural network (FFNN) the signal flows forward from the input layer to the output layer through hidden layers via connections between neurons. The weighted connections are established between neurons in the neighbouring layers, where each neuron is connected to all neurons in the next layer. The neurons are not connected to each other in the same layer [16, 23, 24]. The learning process is iterative, and the network parameters are computed through the update process, where the error in the form of the loss function is minimized by using backpropagation based on the gradient descent algorithm [22, 23]. Various optimization algorithms are available to reduce the error, but the ADAM and Levenberg–

Marquardt algorithms are mostly used [25, 26]. For this training process, a database is required, consisting of input and target output quantities. The database created from experimental investigations is used if available, but the data obtained by computational analyses may also be an option. The FFNN is widely applied in mechanics and materials science to replace conventional constitutive laws, especially for materials with complex heterogeneous microstructure. Furthermore, the FFNN approach can replace the standard homogenization procedure in a representative volume element (RVE) to compute macro state variables within the multiscale numerical algorithms [16]. The modelling of deformation responses of hyperelastic materials by means of the FFNN is presented in [27]. In [28], the FFNN is used to predict the fracture behaviour of particulate composite materials under impact loading. The material fracture modelling based on the phase field approach was efficiently performed by using the FFNN in [29].

To achieve better success in solving complex nonlinear problems, neural networks with multiple hidden layers are required. The training of a neural network with more than three hidden layers is called deep learning (DL) [21, 30]. In DL, the learning is related to a big training database and a large number of computations which require high performance computing and more complex computer architecture, which is permanent in progress. The computers with high GPU (graphics processing unit) capacity are desirable. A special purpose processor for deep learning was also developed [23]. The DL algorithms are highly successful in various fields, such as image recognition, speech recognition, language translation, and biotechnology and medicine sciences, in which case they are related to predicting activities of drug molecules and discovery of new drugs. A historical survey of relevant studies dealing with the DL is presented in [31].

In recent years the DL has been applied in various fields of engineering sciences. The DL is used in the homogenization procedure of a heterogeneous representative volume element within the material modelling based on multiscale algorithms [32]. The study of fracture prediction on the material nanolevel by means of the DL approaches is presented in [33] Deep neural networks are used for damage identification in structural health monitoring in [34]. A great challenge is the application of DL algorithms to the discovery, design, and development of new materials [35, 36, 37]. An efficient DL model is used for the consideration of fracture propagation and failure prediction in brittle materials [38]. The deep learning neural networks are also applied in the finite element technology. The finite element interpolation functions may be generated, and the optimization of nodal positions may be achieved through the training process, which is equivalent to the rh-adaptivity in a standard finite element method (FEM). Higher accuracy and efficiency in comparison to the conventional FEM are achieved, especially in capturing the stress concentration in the case of a coarse mesh [39, 40]. A historical overview of the development of DL algorithms, particularly their applications to the design and behaviour of composite materials is presented in [41]. Although the deep learning has made great advances in accuracy and reliability in recent years, further research is needed to improve its application in complex material systems.

1.2.2 Convolutional neural network

A special type of deep neural networks is the convolutional neural network (CNN), which is usually used in computer vision and applied in fields such as image classification, video recognition, medical image analysis, and self-driving cars among others. Beside the fully connected hidden layers as in the DL networks, the CNN architecture consists of additional hidden layers, such as the convolutional and pooling layers. The input data from the input layer feeds the convolutional layers. In contrast to the DL networks, where neurons from two adjacent layers are connected to each other, in a CNN, only small and localized regions of neurons are connected to a neuron in the next layer. In the convolution layers, the mathematical convolution

operations are included instead of the conventional matrix multiplication used in the fully connected layers. The convolution operation uses kernel functions and can be regarded as a feature identification operation. The network weights are shared, which contributes to the increase in network efficiency. After the convolutional layers, the data are transferred to the pooling layers, where they are compressed and converted into a single vector by means of a flattering procedure. Then, the data are further transferred to a feedforward network based on the fully connected layers [16, 21, 42, 43].

However, the CNN applications are not limited to images. The CNN may also be used in materials science to analyse properties of composite materials with various material distribution [44] as well as microstructural material properties in microstructure images within multiscale modelling [16]. The prediction of material properties of heterogeneous materials by means of the CNN is discussed in [45, 46, 47].

1.2.3 Recurrent neural network

Another class of deep learning models is the recurrent neural network (RNN). In contrast to the CNN described above, which was originally developed to process images and video data, the RNN is designed for processing sequential data, such as speech, text, or other time-series data. While in the traditional deep neural networks the signal flows only forward from input to output which are independent of each other, the learning process is performed over several time steps in the RNN [21, 48, 49]. The output from hidden neurons is not only fed forward to output neurons in the current time step but it is temporarily stored in an internal memory and fed back into the hidden neurons along with the input at the next time step. It means that the neurons in the hidden layer receive both the input at the current time step and the output of the hidden layer generated by the input at the previous time step. An RNN has only a single hidden layer, but it is considered to be a deep network because it can be expressed as a composition of multiple layers when unfolded in time. Accordingly, any RNN can be considered as deep as the sequence is long [48]. However, there are also deep neural networks consisting of multiple recurrent hidden layers, where the learning process is more complex, as described in [21, 50].

An important characteristic of the RNNs is that they share the same parameters within each layer of the network during the sequential processing. Furthermore, they employ the backpropagation through time (BPTT) algorithm in the learning process. The procedure of the BPTT is like that of the standard backpropagation. The difference is that the BPTT adds up errors which propagate through the entire length of the sequence because the same parameters are used, whereas the standard backpropagation does not share the parameters. Consequently, the RNNs suffer from two problems known as the vanishing gradients and the exploding gradients. To solve these problems, more sophisticated versions of the RNN architecture such as the long short-term memory (LSTM) network [51] and the gated recurrent unit (GRU) [52] were proposed. The LSTM reduces the effect of the vanishing gradient by removing the repeated multiplication by the same weight during the backpropagation process. The GRU is based on a concept similar to the LSTM, but it is less complex. The GRU is preferable in the case of small data sets, but the LSTM is generally the most popular.

The RNN methods have been efficiently applied to the history dependent problems in mechanics such as plasticity as well as damage and fracture. The plasticity constitutive laws of microstructural RVEs of heterogeneous materials were found by using the GRU architecture in [53]. The RNN based on a GRU unit is used as a surrogate model of micro-scale simulations in the context of computational multiscale procedure in [54]. A smart constitutive law (SCL) for the homogenization of inelastic microstructure of RVEs was formulated by means of the LSTM in [55], where the loading histories applied to RVEs were expressed by strain sequences. The SCL was implemented into a finite element (FE) solver to model deformation responses of various engineering problems on the macro level. The same SCL was applied to all finite

element integrations points, which is an advantage in comparison to the standard multiscale procedure, where the FE simulation of the microstructure is required for each integration point at each load increment, as discussed in [55]. A new type of recurrent neural network based on the self-consistent concept was proposed for the modelling of elastoplastic solids in [56]. An RNN model using a modified LSTM approach was applied as a surrogate for the micro model within the multiscale FE procedure as shown in [57]. The RNN architecture was efficiently used to predict the fracture evolution in brittle materials in [58]. The brittle fracture was also modelled by using the LSTM networks as shown in [59].

1.2.4 Physics-informed neural network

As evident from the above considerations, the training of neural networks requires a big database, which is not always available for complex scientific problems, and therefore the governing physical laws are mostly ignored. Furthermore, data collection is often very costly and time consuming. Employing the known physical laws in the training process can significantly reduce the size of the required training data as well as increase numerical efficiency. This is particularly so in the case of the modelling of engineering systems governed by their physics. Therefore, a new network architecture called the physics-informed neural network (PINN) is proposed, where the physics constraints are imposed. The PINN may be considered as a special class of the FFNN, where the physical conditions are enforced by a loss function [60, 61].

Like in the FFNN, the weights and bias in the PINN are computed by minimizing a loss function which is given by the residual of governing equations, usually written as partial differential equations (PDEs) and corresponding boundary conditions, respectively. The PDEs are embedded into the loss function using automatic differentiation [62]. This strategy was successfully applied to solving the Burger's equation and the Navier-Stokes equation in the modelling of various phenomena in fluid mechanics in [63]. Two different problems are solved in this study: the differential equation with fixed parameters (data driven solution of PDE) and the parameter identification of differential equations (data-driven discovery of PDE), which may be considered as an inverse process. The application of the PINN in solid mechanics is presented in [64], where the linear elasticity as well as the nonlinear von Mises elastoplasticity are studied. The elastic-viscoplastic deformation in solids in relation to the strain-rate and temperature is modelled in [65]. An advanced PINN technique for modelling two-dimensional and three-dimensional problems in solid mechanics is developed in [66]. Therein, a modified loss function is proposed and both linear and geometrically nonlinear problems are solved. Various types of materials such as the linear elasticity, the large deformation hyperelasticity and the von Mises plasticity are considered in [67], where the meshfree approach in combination with the PINN is applied instead of the classical finite element spatial discretization.

A novel computation technique based on the PINN and applied to heat transfer modelling as well as to solving the inverse problem dealing with computing of the unknown parameters in the PDE is presented in [68]. The heat transfer modelling by means of the PINN is also considered in [69], where the solving of forced and mixed heat transfer problems with unknown thermal boundary conditions as well as of the two-phase problems with a moving interface is demonstrated.

Beside the PINN approach dealing with PDE, the PINN computational strategy is proposed, where the loss function is defined by the energy functional, and the loss is minimized according to the principle of minimum potential energy [70]. Also, the tension, deflection and buckling of an elastic thin square plate are considered. A new PINN algorithm is applied for the phase-field modelling of fracture in [71], where the loss function employs the energy functional consisting of the elastic strain energy and the fracture energy.

Many other computation strategies which may be considered as data driven approaches, where artificial intelligence methods are used, may be found in various fields such as design optimization [72], management [73], and production systems [74], among others.

In the following two sections, the feedforward neural network and the recurrent neural network are considered in more detail. Their applications are demonstrated by two simple examples dealing with elastic and elastoplastic problems in solid mechanics. Finally, concluding remarks and possible future work directions are presented in Section 4.

2. Feedforward neural network

2.1 Feedforward neural network formulation

As mentioned in the previous section, the feedforward neural network (FFNN) represents the fundamental network architecture which consists of network layers. There are one input, one output and multiple hidden layers. Each layer contains multiple neurons. The signal flows from each input neuron to each neuron of the first hidden layer and it flows further forward to the neurons of the final output layer through connections between the neurons of other hidden layers. Hence, each neuron receives the input signal from all neurons of the preceding layer, which is their output. The output is computed by means of the activation function in terms of the input value. Each connection between the neurons is characterized by its weight parameter. The bias is another parameter which is associated to the neurons. A multi-layered feedforward neural network with the same number of neurons in each hidden layers is shown in Fig. 1. It is to note that the layers may contain different number of neurons in general.

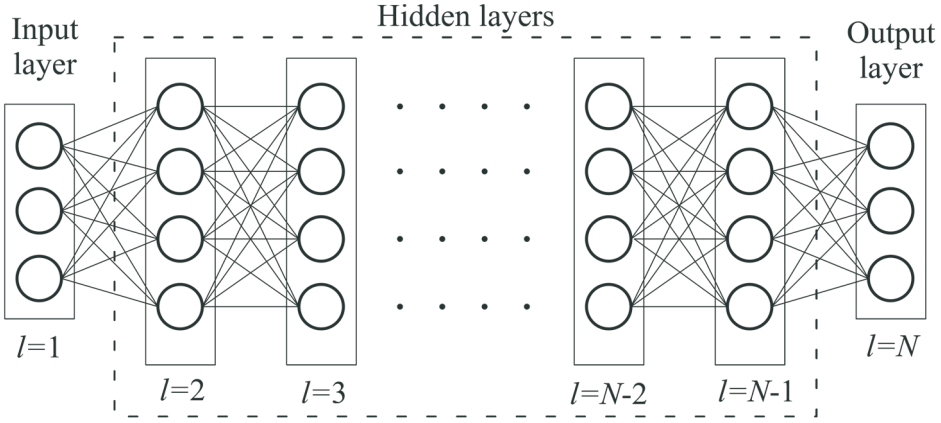

Fig. 1 Feedforward neural network with multiple layers

The input value to the activation function of the *j*-th neuron in the *l*-th layer is computed by the following relation:

$$u_j^l = \sum_{i=1}^{n^{l-1}} w_{ji}^{l-1} o_i^{l-1} + b_j^l, \tag{1}$$

where w_{ji}^{l-1} is the connection weight between the *i*-th neuron in the $(l-1)$-th layer and *j*-th neuron in the *l*-th layer. The value o_i^{l-1} is the output of the *i*-th neuron in the $(l-1)$-th layer. b_j^l is the bias of the *j*-th neuron in the *l*-th layer. The indices *i* and *j* represent neurons in the previous and current layer, respectively, and *n* is the number of neurons in layers, $i = 1\ldots n$, $j = 1\ldots n$, while *N* is the number of layers, $l = 1\ldots N$. It should be noted that *i* is an index that is summed over according to the Einstein summation convention. In the architectural graph in Fig. 1, the neurons are illustrated by nodes (small circles) which are connected to each other by straight

lines. There are three source nodes, *N*-2 hidden layers of four neurons each and three output neurons. In that case, the input of the *j*-th neuron of the *N*-th output layer may be written as

$$u_j^N = w_{ji}^{N-1} o_i^{N-1} + b_j^N, \quad i = 1...4. \tag{2}$$

The output of the *j*-th neuron in the *N*-th layer is expressed by an activation function $f(\,)$ as

$$o_j^N = f(u_j^N). \tag{3}$$

There are various activation functions, but the nonlinear sigmoid function and the hyperbolic tangent (tanh) expressed by

$$f(x) = \frac{1}{1+e^{-x}} \quad \text{and} \quad f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{4}$$

respectively, are usually employed in computational mechanics. In addition, the following linear functions may also find application, such as the rectified linear unit (ReLU) written as

$$f(x) = \begin{cases} x \ (x > 0) \\ 0 \ (x \le 0) \end{cases} \tag{5}$$

and the identity function $f(x) = x$. Besides, there are several activation functions which may be employed in neural networks, as described in [22]. The functions ReLU (*x*), sigmoid (x) and tanh (x) are presented in Fig. 2.

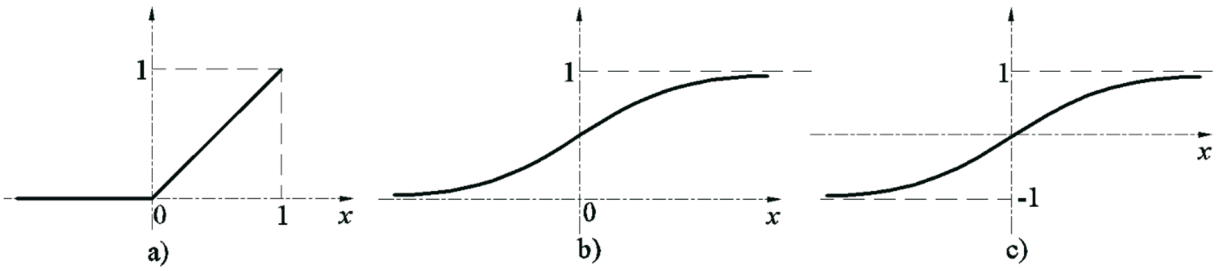

Fig. 2 Activation functions: a) ReLU (*x*), b) sigmoid (*x*), c) tanh(*x*)

For the learning process started with the input layer, training data consisting of inputs and target outputs are needed. The data may be collected from experimental or computational analyses. A pair of the input and corresponding output data is called a training pattern [23]. The collected data are generally divided into three different sets, i.e., training, validation, and testing sets. The training set comprises approximately 70% or 80% data of the complete data, while the validation and testing contain 15% or 10% data each. The data are selected at random from the full data set. The amount of the data used for the learning depends on the complexity of the problems which should be solved. The size of the training data is related to the neural network architecture. Both the size of the data set and the complexity of the problem to be solved influence the number of neurons and layers as well as the type of activation functions.

For simple problems, the networks with two or three hidden layers should be enough, but deep neural networks employing multiple layers are required for solving complex problems. Unfortunately, there are no uniform rules on how to select the network architecture. If too many neurons are used, the undesired phenomena such as overfitting may appear, which can be prevented by early stopping or by means of regularization procedures. The Bayesian regularization is usually used [22]. The simplest choice is the size of the output layer, which is the same as the number of the target data. On the other hand, the choice of the input layer may be difficult, especially in the case when the potential number of input data is too large and some

reducing techniques are required. To ensure efficiency and accuracy of the learning, it is suggested to normalize the input data so that they fall into the unit range from -1 to 1 or 0 to 1. In that case, undesired major differences between the computed values through the learning process may be avoided. Furthermore, using the normalized data improves the learning process because the weight parameters and biases do not have large values.

During the learning process, the network parameters such as weight and bias are iteratively computed through the update procedure, where the error expressed by the loss function is minimized. The back propagation based on the gradient descent algorithm is employed. Using the calculated output data and the corresponding given target data, the loss function as the mean squared error is computed by the following expression:

$$E = \frac{1}{2}\sum_{s=1}^{n_s}\sum_{j=1}^{n}({}^{s}o_j - {}^{s}o_j^*)^2 , \tag{6}$$

where ${}^{s}o_j$ is the output of the *j*-th neuron in the output layer for the *s*-th training pattern. The value ${}^{s}o_j^*$ denotes the given target data corresponding to the output of the *j*-th neuron in the output layer for the *s*-th training pattern. *n* is the total number of neurons at the output layer and n_s stands for the total number of the training pattern.

According to the gradient descent concept, the change of the connection weight between the *j*-th neuron in the *l*-th layer and the *i*-th neuron in the $(l-1)$-th layer is expressed by

$$\begin{aligned} \Delta w_{ji}^{l-1} &= \frac{\partial E}{\partial w_{ji}^{l-1}} , \\ w_{ji} &= w_{ji}^{l-1} - \alpha \Delta w_{ji}^{l-1} . \end{aligned} \tag{7}$$

The change of the bias is performed analogously:

$$\begin{aligned} \Delta b_j^l &= \frac{\partial E}{\partial b_j^l} , \\ b_j^l &= b_j^l - \beta \Delta b_j^l . \end{aligned} \tag{8}$$

In the above relations, α and β are the parameters which define the step size. In the training algorithm, after the forward procedure and after computing the loss function, the backpropagation follows, where the connection weights and the biases are updated from the output layer to the input layer by using expressions (7) and (8). This gradient descent process may be considered as an optimization approach in which the weights and biases converge to their optimal values for which the minimal error expressed by (6) is reached. The backpropagation algorithm starts after the forward computations are performed for all training patterns. The sum of forward computations for all training patterns is called epoch [23]. Accordingly, the error expressed by Eq. (6) gradually decreases epoch by epoch. Various optimization approaches are available, but the ADAM and Levenberg–Marquardt algorithms are mostly employed [22, 25]. Before the updating process, the weights and biases should be initialized, and the stopping criteria should be defined [22]. The initialization method depends on the network architecture. One possibility is to randomly select the weight and bias values so that they fall within the range of normalized input values. The stopping criteria may be to stop the training when the error reaches its specified limit or when the performance on the validation set increases by a set number of iterations [22].

2.2 Application of FFNNs to computation of an elastic problem

In this section, a simple application of a feedforward neural network (FFNN) is presented by solving an elastic boundary value problem. A porous square plate shown in Fig. 3 subjected to the periodic boundary conditions expressed by the displacement along the sides is considered. Due to the strain imposed, the homogenized stress is to be computed. The side length is l = 200 mm, and the plate is discretized by 1,198 quadrilateral CPE4 finite elements [75]. The voids are randomly distributed over the plate, which yields 30% of porosity. This example may also represent a heterogeneous microstructure within the multiscale modelling of structural deformation responses [76]. The finite element computations were performed using the finite element program *Abaqus* [75].

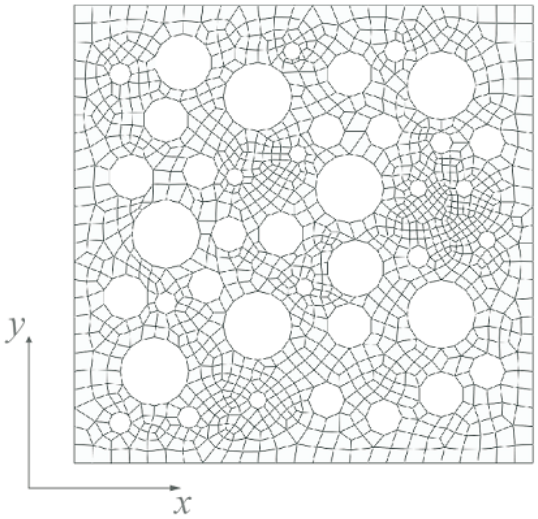

Fig. 3 Discretized porous square plate

The elastic behaviour of the plate is described by Young's modulus E = 210000 MPa and Poisson's ratio υ = 0.3. To obtain the necessary data for training the neural network, the heterogeneous plate is subjected to a series of plane strain loading conditions, which are used to generate a database of strain and homogenized stress values. In the present study, there are seven different loading conditions, each with 100 increments, representing different combinations of the strain components $\varepsilon_x, \varepsilon_y, \gamma_{xy}$ applied to the plate. Three loading conditions consist of just one strain component and the remaining four loading cases are the combinations of multiple strain components. The strain components used in all loading scenarios are up to the value of 2%. The strains are transformed into the boundary displacements which are periodically imposed along the boundary nodes. Accordingly, the boundary nodes displacement vector is expressed as

$$\mathbf{u}_{\mathrm{b}} = \mathbf{D}^T \boldsymbol{\varepsilon}\,, \tag{9}$$

where $\boldsymbol{\varepsilon}$ denotes the strain vector and $\mathbf{D}$ is the coordinate matrix which may be expressed by the submatrices $\mathbf{D}_i$ containing the boundary nodes coordinates, and may be written for n nodes as

$$\mathbf{D} = \left[\mathbf{D}_1\ \mathbf{D}_2 \cdots \mathbf{D}_n\right]. \tag{10}$$

Here, the matrix for boundary node i has the following form:

$$\mathbf{D}_i = \frac{1}{2}\begin{bmatrix} 2x & 0 & y \\ 0 & 2y & x \end{bmatrix}. \tag{11}$$

More information about periodic boundary conditions can be found in [76]. Once the boundary value problem of the porous square plate is solved, the homogenized stress vector is calculated using the equation

$$\boldsymbol{\sigma} = \frac{1}{V}\int_V \boldsymbol{\sigma}_e \mathrm{d}V\,, \tag{12}$$

where $\boldsymbol{\sigma}_e$ is the stress vector in each element material point. According to the plane strain assumption, the homogenized stress vector contains three normal components σ_x, σ_y, σ_z and one shear component τ_{xy}. For each of seven loading cases, the finite element calculation results in a database of 700 training patterns (data points), each consisting of three strain components (ε_x, ε_y, γ_{xy}) and four homogenized stress components (σ_x, σ_y, σ_z, τ_{xy}). The obtained data is then used to train the neural network.

In the case of the described problem, a FFNN with one hidden layer and five neurons is used. The input layer of the network has three neurons, which correspond to the three strain components, while the output layer has four neurons, corresponding to the four homogenized stress components. The choice of the single hidden layer is made based on the observation that for the linear elastic mechanical problem, the single hidden layer is sufficient to achieve high accuracy. The activation function is the previously mentioned identity function, which is a common choice for problems involving linear responses. The neural network is made by using *TensorFlow* and its *Keras* API (application programming interface). *TensorFlow* is an open-source software library developed by *Google* for numerical computations and machine learning. It is designed to enable efficient computation and large-scale distributed training of neural networks. It includes a high-level API called *Keras*, which makes easy to build and train deep neural networks. *Keras* provides a simple and intuitive interface for defining neural network models, allowing users to focus on the high-level architecture of their models without worrying too much about the low-level details of *TensorFlow*. More information about *TensorFlow* and *Keras* can be found in [77, 78]. The schematic of the used FFNN is shown in Fig. 4.

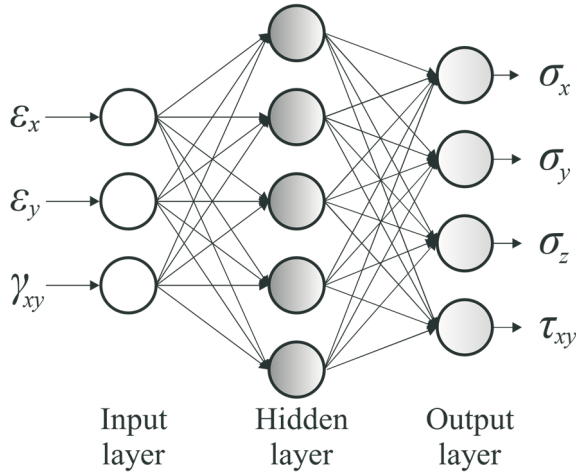

Fig. 4 Schematic of the used FFNN

As described in the previous section, the collected data are normalized to range from -1 to 1 to improve the convergence of the training process. The normalization is performed for every strain and stress component individually, by using the following equation:

$$x_{\mathrm{scl}} = 2\cdot\frac{x - x_{\min}}{x_{\max} - x_{\min}} - 1\,, \tag{13}$$

where $x_{\min}$ and $x_{\max}$ are the lowest and the highest values of the components, respectively. Furthermore, all data are randomly shuffled and divided in the training and the validation set. 80% of the data are used for the training, and the remaining 20% are set aside for the validation. The use of a validation data set is important to prevent the well-known overfitting effect.

The training of the neural network is performed for 100 epochs by using the mean squared error as the loss function and the ADAM optimizer. One epoch corresponds to the period in which the forward and backpropagation calculations are done for all training patterns. The choice of using the mean squared error (MSE) as the loss function and ADAM as the optimizer for the network training is motivated by its simplicity and effectiveness. The ADAM algorithm is designed to overcome some of the limitations of the standard stochastic gradient descent (SGD) algorithm by introducing a few key innovations. One of the main innovations of the ADAM algorithm is the use of adaptive learning rates that are based on estimates of the first and the second moments of the gradients. This means, the learning rate is adjusted automatically based on the gradient history, allowing the algorithm to converge faster and more reliably. Another key innovation of the ADAM algorithm is the use of the momentum. The momentum is a technique that helps the algorithm to continue making progress even when the gradients are small or noisy. More information about ADAM and detailed description of the weight and bias parameters optimization can be found in [79]. In the past, the neural network weights were typically initialized with small random values. Nowadays, the advanced deep learning libraries like *Keras* provide a variety of network initialization techniques, which all involve initializing weights with small random numbers in some way or another. The default initializer, used also in this example, is *Glorot uniform* and more information about it can be found in [80].

The value changes of the resulting loss function during the epochs are shown in Fig. 5. As evident, the loss values rapidly decrease during the first 40 epochs, and then they decrease only slightly until the end of the training process. The validation loss value does not increase, and it is not significantly greater than the loss value of the training data set, therefore, it shows that the overfitting does not occur.

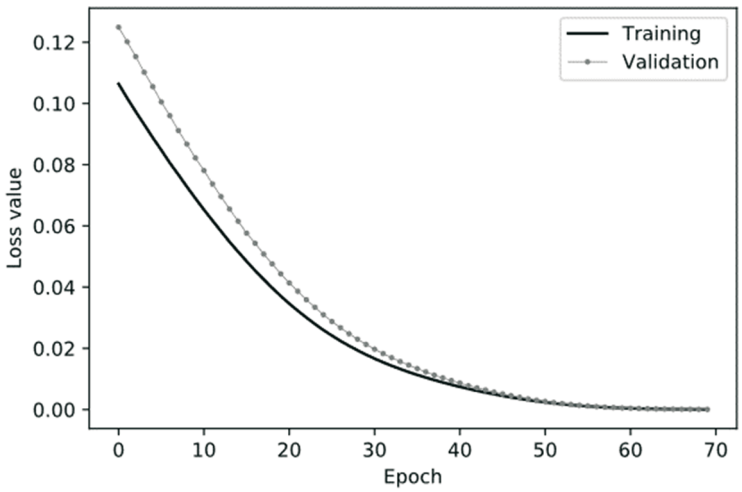

Fig. 5 Loss value changes over epochs

Once the neural network is trained, it is tested on a load case with the strain components ε_x, ε_y up to 1.5% and 1%, respectively, and the shear strain γ_{xy} of up to 1.4%. The neural network's prediction is compared to the *Abaqus* FEM results, and it is found to be accurate. The comparison of the homogenized normal stress in the *x* direction obtained by using both the

FFNN and *Abaqus* are shown in Fig. 6. All other homogenized stress components show the same matching.

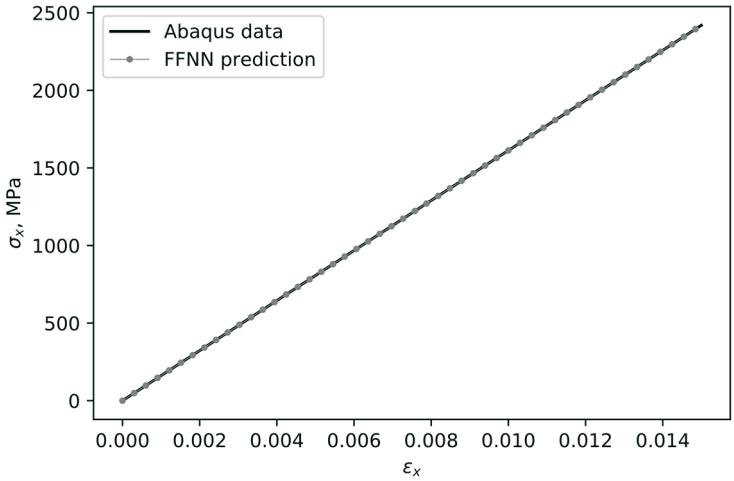

Fig. 6 FFNN stress-strain diagram

One of the most important advantages of using neural networks for calculation of the homogenized stress is the required CPU time as shown in Table 2.1. The calculation of the homogenized stress state using *Abaqus* takes 9.9 seconds, while the same can be obtained by using neural networks in just 0.109 seconds. It should be pointed out that *Abaqus* performs far more tasks than just the FEM calculation, but it is obvious that the coupling neural network prediction with the *Abaqus* calculation process can lead to its acceleration. However, the use of the neural network prediction is limited to the model and cases the neural network is trained on. The use of neural networks in a general approach would require training on many different models and loading cases.

Table 1 FFNN CPU time comparison

	CPU time [s]
Abaqus FEM simulation	9.9
FFNN prediction	0.109

3. Recurrent neural network

3.1 Recurrent neural network architecture

As mentioned in the introductory section, the recurrent neural network (RNN) deals with sequential data, where the learning process is performed over several time steps. The input data are time dependent, and besides the data of the current time step, the data of the previous time step are also considered. A standard RNN has only a single hidden layer, into which a working memory is included. In the current time step the information flows not only forward, but it is also temporarily stored in the working memory and then it flows into the hidden neurons along with the input at the next time step. In this way, the neurons in the hidden layer receive both the input at the current time step and the output of the hidden layer generated by the input at the previous time step. The signal from the working memory propagates through time, and therefore the RNN may be considered as deep neural network. The time-unrolled computational graph presenting the architecture of an RNN is shown in Fig. 7. However, the RNN may also

be graphically presented more compact as a rolled computational graph, where the sequential data processing is presented in the form of loops [21].

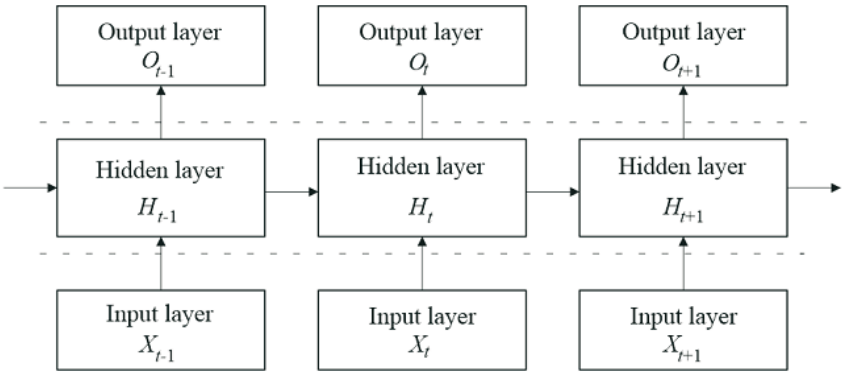

Fig. 7 Time-unrolled RNN computational graph

As evident from Fig. 7, there is a sequence of inputs represented by the layers X_{t-1}, X_t, X_{t+1}, and each of them is mapped on the corresponding output (O_{t-1}, O_t, O_{t+1}) through the hidden layers H_{t-1}, H_t, H_{t+1}, where the connections between the layers are represented by the vertical arrows. Each of these connections is parametrized by the weights in the same way as for the feedforward procedure described in the previous section. The information stored in the memory of each hidden layer propagates from one time step to the next time step or from one hidden layer to the next hidden layer, as shown by the horizontal arrows. These connections are also parametrized by the weights. In this network architecture, the activation functions and the biases are also used analogously to the standard feedforward approach. It should be stressed that the same parameters are shared during the sequential processing, which means that the same parameters are used at every time step. All the activation functions may also be the same. Different design patterns of recurrent neural networks are also available as shown in [Goodfellow 2016], but the architecture presented above is mostly used. As mentioned in the introductory section, the RNNs employing multiple hidden layers, called deep recurrent networks, were also proposed [21, 50].

Analogously to the feedforward approach, the learning process employing an iterative update procedure using the back propagation based on the gradient descent algorithm is performed. Here, the backpropagation through time (BPTT) algorithm [21] is applied. Since the RNN is based on the mapping of an input sequence on an output sequence, the total loss function is computed as the sum of the losses over all the time steps. The gradient of the loss function with respect to the network parameters such as the weight and bias should be computed for each neuron. During the learning process the error should backpropagate through the entire length of sequence, and it is used to update the weights on the connections. The error is multiplied by the connection weights between the hidden layers, where the weights are stationary through time, and they do not change from one time step to the next time step. Thus, the backpropagation through multiple time steps implicates the repeated multiplication of the error gradient by the same set of weights. If this weight is a small value, it may diminish at an exponential rate, and the error gradient also tends to diminish. Consequently, the gradient descent will never converge to an optimum, and this problem is called the vanishing gradient [48]. To solve this problem, long short-term memory networks (LSTMs) are proposed.

3.2 Long short-term memory networks

In contrast to the standard RNNs, the LSTMs have an additional component in their structure which is called the cell, where the information is maintained and propagated forward through time. The flow of information within the cell is regulated by the structural components called the gates, which are set in the LSTM hidden layer units. There are the forget gate, the input gate, and the output gate. Each gate consists of layers of neurons, which contain one neuron per activation function in the cell state. The information coming into an LSTM hidden layer unit includes the cell state from the previous time step c_{t-1}, the working memory from the previous time step H_{t-1} and the input value from the current time step X_t, as shown in Fig. 8.

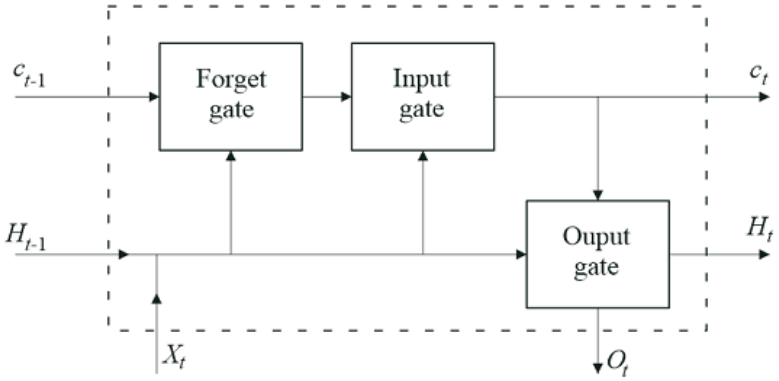

Fig. 8 Schematic of internal structure of LSTM unit

The forget gate has the role to decide what information in the cell should be forgotten at each time step. The sigmoid activation functions are used, and their output numbers are between zero and one. Since the cell state is multiplied by these numbers, the information multiplied by zero is forgotten, and the other piece of information multiplied by one is remembered. The input gate decides what information should be stored in the cell state and it is divided into two parts. In the first part, the sigmoid activation functions are again used to determine the values in the cell state to be updated. The second part uses the tanh activation function with the output values between -1 and +1 and it identifies what information should be updated. Finally, the results of both parts are multiplied by each other and added to the cell state. Thus, the old cell state is updated into the new state c_t. The third gate, the output, decides which values should be output in response to the current input. Both the tanh and the sigmoid activation functions are employed. Now, the output of the LSTM hidden layer unit consists of the cell state from the current time step c_t, and H_t and O_t stand for the working memory and the output from the current time step, respectively, as shown in Fig. 8. More detailed information on LSTMs can be found in [48, 81].

3.3 Application of the LSTM neural network to an elastoplastic problem

The purpose of this section is to show the use of the LSTM recurrent neural network in the computation of an elastoplastic problem. The same porous plate as in the previous example is considered. The material data describing elastic behaviour are also unchanged. The plastic response is defined by the initial yield stress of 250 MPa and the linear isotropic hardening is described by the plastic modulus of 50 GPa. Subjecting a porous plate to a series of plane strain loading conditions is done in the same manner as in the example presented in the previous section. Since the elastoplastic problem is a more complicated issue than the computation of

the elastic response, the number of different loading conditions must be increased. Hence, a total number of 100 different combinations of the strain components (ε_x, ε_y, γ_{xy}) are applied using the periodic boundary conditions expressed by equation (9) to obtain an adequate database needed for the LSTM training.

The nonlinear finite element (FE) analysis has been performed by means of the linearized finite element equation $\mathbf{K}\Delta\mathbf{v} = \mathbf{F}_\mathrm{e} - \mathbf{F}_\mathrm{i}$, where $\mathbf{K}$ is the tangent stiffness matrix employing the consistent elastoplastic tangent modulus, $\Delta\mathbf{v}$ is the increment of the nodal displacement vector, and $\mathbf{F}_\mathrm{e}$ and $\mathbf{F}_\mathrm{i}$ are the external and internal nodal force vectors, respectively. The finite element equation is solved by using iterative procedures. The FE computations were performed using the FE program *Abaqus* [75]. The elastoplastic algorithm employs the isotropic hardening von Mises model. The integration of the nonlinear constitutive equation was performed by the return mapping algorithm. The basic equations of the elastoplastic analysis are shown in Table 2. More details on elastoplastic computations can be found in [82, 83].

Table 2 Basic equations of elastoplastic analysis

Additive decomposition of strain tensor: $\mathrm{d}\varepsilon_{ij} = \mathrm{d}\varepsilon_{ij}^{e} + \mathrm{d}\varepsilon_{ij}^{p}$
Stress increment: $\mathrm{d}\sigma_{ij} = D_{ijkl}\left(\mathrm{d}\varepsilon_{kl} - \mathrm{d}\varepsilon_{kl}^{p}\right)$
Plastic strain increment: $\mathrm{d}\varepsilon_{ij}^{p} = \mathrm{d}\lambda\dfrac{\partial f}{\partial\sigma_{ij}}$
Von Mises yield surface: $f\left(\sigma_{ij},\varepsilon_{ij}^{p}\right) = J_2 - \dfrac{1}{3}\sigma_Y^2 = 0$
Linear isotropic hardening: $\sigma_Y = \sigma_{Yo} + K\varepsilon_{eq}^{p}$
Effective plastic strain increment: $\mathrm{d}\varepsilon_{eq}^{p} = \sqrt{\dfrac{2}{3}\mathrm{d}\varepsilon_{ij}^{p}\,\mathrm{d}\varepsilon_{ij}^{p}}$
Kuhn-Tucker complementarity conditions: $f\left(\sigma_{ij},\varepsilon_{ij}^{p}\right)\le 0,\quad \lambda\ge 0,\quad \lambda f\left(\sigma_{ij},\varepsilon_{ij}^{p}\right) = 0$

In the relation presented in Table 2, D_{ijkl} is the elastic stiffness tensor, λ is a scalar plastic multiplier, J_2 denotes the second invariant of the stress deviator tensor, σ_{Yo} stands for the initial yield stress, and K represents the plastic hardening modulus.

After running one hundred finite element simulations, a database of strain-stress pairs for each loading condition is obtained. As in the previous example, the input data are three strain components (ε_x, ε_y, γ_{xy}) and the output consists of four homogenized stress components (σ_x, σ_y, σ_z, τ_{xy}). This database is used to train an LSTM recurrent neural network for predicting the stress components from the input sets of the strain components. The database consists of 100 loading samples, each 100 time steps long and each time step contains all the strain and homogenized stress components.

The created neural network consists of two hidden layers, each with 120 LSTM units. The input layer again consists of three neurons (for three strain components), while the output layer has four neurons (for four stress components). The neural network is made by using *TensorFlow* and its *Keras* API, already described in the previous section. To prepare the data

for the training, it is normalized to range from -1 to 1 by means of expression (11). As mentioned before, the data set is again divided into two parts, 80% for training and 20% for validation.

The network is trained for 220 epochs, with the loss function expressed by the mean squared error and the ADAM algorithm used as the optimizer in combination with the backpropagation through time (BPTT). The BPTT algorithm can be described as follows:

1. provide a sequence of input and output pairs,
2. unroll the network to cover all the time steps and accumulate errors for each time step,
3. roll-up the network and adjust the weights based on the calculated errors and the ADAM algorithm [84].

Moreover, the early stopping criteria with patience of 30 epochs is used. The patience of 30 epochs means that if the loss function value for the validation data set does not decrease in that period, the training is stopped, and the weights and biases of the epoch with the lowest loss are used for prediction. The epoch now corresponds to the period until the forward and backward propagations are done for all time steps of all 100 loading conditions. The neural network weights and biases are initialized with the *Glorot uniform* technique, which is also used in the previous example. As shown in Fig. 9, the loss value for both the training and the validation data sets has greatly decreased in the first 50 epochs, and the slight decline in the loss value continued until the epoch number 196. After this epoch, the loss value for the validation data set does not longer decrease, so the early stopping criteria is activated after 30 epochs. The weights and biases of the epoch 196 are saved and used for predictions.

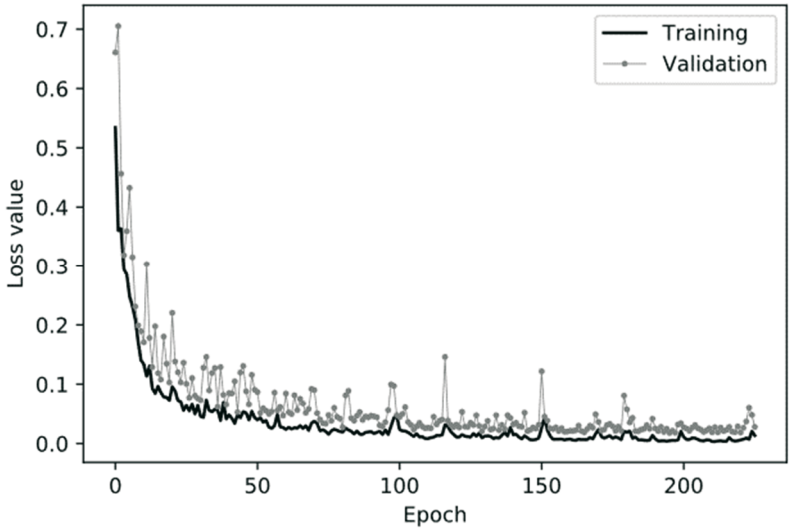

Fig. 9 Loss value changes over epochs

The trained neural network is tested on the load cases expressed by the strain components ε_x up to 3%, ε_y up to 2% and the shear strain component γ_{xy} of 3%. Again, the prediction of the trained neural network is compared to the results obtained by the *Abaqus* FEM analysis. As can be seen in Fig. 10, the LSTM neural network prediction of the homogenized equivalent von Mises stress shows high accuracy.

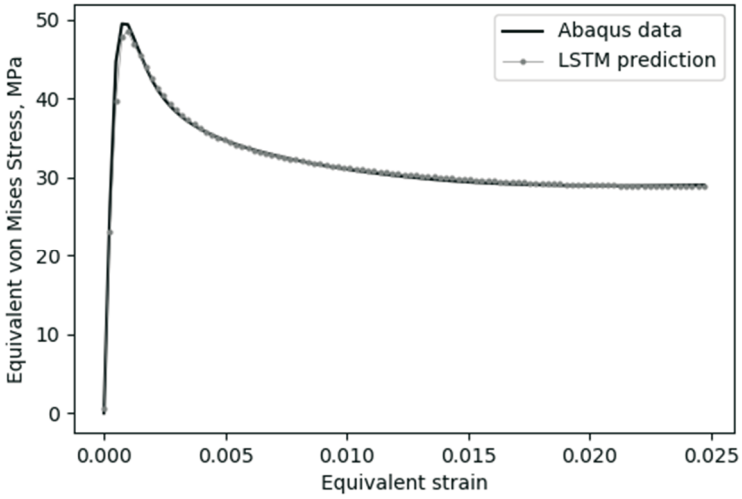

Fig. 10 Equivalent von Mises stress-strain diagram

The comparison of the CPU time needed for the *Abaqus* analysis and the LSTM prediction is presented in Table 3 showing a great computational potential of the LSTM usage in solid mechanics analyses. As it is stated in Section 2.2, the incorporation of the neural network into *Abaqus* could increase its calculation speed. On the other hand, the above shown neural network can only be used on the model and loading cases it is trained on, which is an obvious limitation to its usage.

Table 3 LSTM CPU time comparison

	CPU time [s]
Abaqus FEM simulation	15.5
LSTM prediction	0.25

4. Conclusions

A review of the machine learning approaches within the artificial intelligence methods with a special emphasis on the neural network techniques is presented. The mostly used neural network architectures, such as the feedforward neural network, the convolutional neural network, the recurrent neural network, and the physics-informed neural network, are discussed. A special attention is directed to their application in the field of solid mechanics. The feedforward neural network and the recurrent neural network are described in more detail in separate sections. To solve the vanishing gradient problem within the recurrent neural architecture, the long short-term memory (LSTM) network is presented.

Two simple examples of neural network application are shown. In the first example, the feedforward neural network is used for solving an elastic boundary value problem, where a porous plate subjected to the periodic boundary condition is considered. In the second example, the LSTM recurrent neural network is applied to solve an elastoplastic boundary value problem of the same plate. Both the elastic and the elastoplastic computations were also performed by means of the finite element method using the software package *Abaqus*, and the results and the numerical efficiency are compared. The neural network approaches show lower computational cost when compared to the finite element computations, while high accuracy is preserved.

However, the design of the neural network architecture is still a great challenge, and it is an extremely active area of research especially in the field of solid mechanics. Guiding theoretical principles have not been formulated yet, and the design usually includes the process of trial and error. The formulation of an efficient loss function is still an open issue. An advance on neural network procedures and applied algorithms are expected in the future.

Acknowledgements

This study has been supported and co-funded by the European Union through the European Regional Development Fund, Operational Programme "Competitiveness and Cohesion 2014 – 2020" of the Republic of Croatia, project ImproWE - Improvement of High-Efficiency Welding Technology (KK.01.1.1.07.0075).

REFERENCES

[1] Haykin S. Neural Networks. A Comprehensive Foundation. Prentice Hall; 1999

[2] Mitchell TM. Machine Learning. McGraw-Hill Series in Computer Science. New York: McGraw-Hill; 1997.

[3] Butler KT, Davies DW, Cartwright H, Isayev O, Walsh A. Machine learning for molecular and materials science. Nature. 2018;559(7715):547-55. https://doi.org/10.1038/s41586-018-0337-2

[4] Akil H, Martone ME, Van Essen DC. Challenges and Opportunities in Mining Neuroscience Data. Science. 2011;331(6018):708-12. https://doi.org/10.1126/science.1199305

[5] Bellazzi R, Zupan B. Predictive data mining in clinical medicine: Current issues and guidelines. International Journal of Medical Informatics. 2008;77(2):81-97. https://doi.org/10.1016/j.ijmedinf.2006.11.006

[6] Dara S, Dhamercherla S, Jadav SS, Babu CM, Ahsan MJ. Machine Learning in Drug Discovery: A Review. Artificial Intelligence Review. 2022;55:1947-1999. https://doi.org/10.1007/s10462-021-10058-4

[7] Bianchini M, Simic M, Ghosh A, Shaw RN. Machine Learning for Robotics Applications. 1st ed. Singapore, Singapore: Springer; 2022. https://doi.org/10.1007/978-981-16-0598-7

[8] Jain M, Marasini B, Jha AK, Thapa P, Jasbir. Autonomous Vehicle Using Various Machine Learning Algorithms. International Journal of Scientific Research in Computer Science, Engineering and Information Technology 2019;5:754-760. https://doi.org/10.32628/CSEIT1952205

[9] Wäschle M, Thaler F, Berres A, Pölzlbauer F, Albers A. A review on AI Safety in highly automated driving. Front Artif Intell. 2022;5:952773. https://doi.org/10.3389/frai.2022.952773

[10] Nath RR, Kakoty K, Bora DJ. Face Detection and Recognition Using Machine Learning Algorithm. UGC Care Journal 2020;43:194-197

[11] Sharma S, Bhatt M, Sharma P. Face recognition system using machine learning algorithm. In: 2020 5th International Conference on Communication and Electronics Systems (ICCES). IEEE; 2020. https://doi.org/10.1109/ICCES48766.2020.9137850

[12] Krishnan CG, Robinson YH, Chilamkurti N. Machine Learning Techniques for Speech Recognition using the Magnitude. Journal of Multimedia Information System 2020;7(1):33-40. https://doi.org/10.33851/JMIS.2020.7.1.33

[13] Vashisht V, Pandey AK, Yadav SP. Speech Recognition using Machine Learning. IEIE Transactions on Smart Processing and Computing 202;10:233-239. https://doi.org/10.5573/IEIESPC.2021.10.3.233

[14] Kirchdoerfer T, Ortiz M. Data-driven computational mechanics. Computer Methods in Applied Mechanics and Engineering. 2016;304:81-101. https://doi.org/10.1016/j.cma.2016.02.001

[15] Ibanez R, Borzacchiello D, Aguado JV, Abisset-Chavanne E, Cueto E, Ladeveze P, Chinesta F. Data-driven non-linear elasticity: constitutive manifold construction and problem discretization. Computational Mechanics 2017;60:813-826. https://doi.org/10.1007/s00466-017-1440-1

[16] Li H, Kafka OL, Gao J, Yu C, Nie Y, Zhang L, et al. Clustering discretization methods for generation of material performance databases in machine learning and design optimization. Computational Mechanics. 2019 May 22;64(2):281-305. https://doi.org/10.1007/s00466-019-01716-0

[17] Schmidt J, Marques MRG, Botti S, Marques MAL. Recent advances and applications of machine learning in solid-state materials science. Computational Materials 2019;5(1):1-36. https://doi.org/10.1038/s41524-019-0221-0

[18] Wagner N, Rondinelli JM. Theory-Guided Machine Learning in Materials Science. Frontiers in Materials. 2016;3:1-9. https://doi.org/10.3389/fmats.2016.00028

[19] Bessa MA, Bostanabad R, Liu Z, Hu A, Apley DW, Brinson C, et al. A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality. Computer Methods in Applied Mechanics and Engineering 2017;320:633-67. https://doi.org/10.1016/j.cma.2017.03.037

[20] Lei X, Liu C, Du Z, Zhang W, Guo X. Machine Learning-Driven Real-Time Topology Optimization Under Moving Morphable Component-Based Framework. Journal of Applied Mechanics. 2018;86(1). https://doi.org/10.1115/1.4041319

[21] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT Press. The MIT Press, Massachusetts Institute of Technology; 2021

[22] Hagan MT, Demuth HB, Beale MH. Neural Network Design (2nd Edition). 2nd ed. Martin Hagan; 2014.

[23] Yagawa G, Oishi A. Computational mechanics with neural networks. 1st ed. Cham, Switzerland: Springer Nature; 2021. https://doi.org/10.1007/978-3-030-66111-3

[24] Oishi A, Yagawa G. Computational mechanics enhanced by deep learning. Computer Methods in Applied Mechanics and Engineering. 2017;327:327-51. https://doi.org/10.1016/j.cma.2017.08.040

[25] Chilimbi T, Suzue Y, Apacible J, Kalyanaraman K. Project Adam: Building an Efficient and Scalable Deep Learning Training System, In 2014: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation; 2014.

[26] Hagan MT, Menhaj MB. Training feedforward networks with the Marquardt algorithm. IEEE Transactions on Neural Networks. 1994;5(6):989-93. https://doi.org/10.1109/72.329697

[27] Shen Y, Chandrashekhara K, Breig WF, Oliver LR. Neural Network Based Constitutive Model for Rubber Material. Rubber Chemistry and Technology. 2004;77(2):257-77. https://doi.org/10.5254/1.3547822

[28] Kushvaha V, Kumar SA, Madhushri P, Sharma A. Artificial neural network technique to predict dynamic fracture of particulate composite. Journal of Composite Materials. 2020;54(22):3099-108. https://doi.org/10.1177/0021998320911418

[29] Aldakheel F, Satari R, Wriggers P. Feed-forward neural networks for failure mechanics problems. Applied Sciences 2021;11(14):1-22. https://doi.org/10.3390/app11146483

[30] LeCun Y, Bengio Y, Hinton G. Deep Learning. Nature. 2015;521(7553):436-44. https://doi.org/10.1038/nature14539

[31] Schmidhuber J. Deep learning in neural networks: An overview. Neural Networks. 2015;61:85-117. https://doi.org/10.1016/j.neunet.2014.09.003

[32] Liu Z, Wu CT, Koishi M. A deep material network for multiscale topology learning and accelerated nonlinear modeling of heterogeneous materials. Computer Methods in Applied Mechanics and Engineering. 2019;345:1138-68. https://doi.org/10.1016/j.cma.2018.09.020

[33] Lew AJ, Yu CH, Hsu YC, Buehler MJ. Deep learning model to predict fracture mechanisms of graphene. 2D Materials and Applications. 2021;5(1). https://doi.org/10.1038/s41699-021-00228-x

[34] Betancourt D, Freitag S, Muhanna RL. Damage detection for structural health monitoring under uncertainty using deep interval neural networks. In 2021: Proceedings of International Workshop on Reliable Engineering Computing (REC2021) 2021;.9:1-16

[35] Kim Y, Kim Y, Yang C, Park K, Gu GX, Ryu S. Deep learning framework for material design space exploration using active transfer learning and data augmentation. Computational Materials. 2021;7(1). https://doi.org/10.1038/s41524-021-00609-2

[36] Agrawal A, Choudhary A. Deep materials informatics: Applications of deep learning in materials science. MRS Communications. 2019;1-14. https://doi.org/10.1557/mrc.2019.73

[37] Linka K, Hillgärtner M, Abdolazizi KP, Aydin RC, Itskov M, Cyron CJ. Constitutive artificial neural networks: A fast and general approach to predictive data-driven constitutive modeling by deep learning. Journal of Computational Physics. 2021;429:110010. https://doi.org/10.1016/j.jcp.2020.110010

[38] Wang Y, Oyen D, Guo W (Grace), Mehta A, Scott CB, Panda N, et al. StressNet - Deep learning to predict stress with fracture propagation in brittle materials. Materials Degradation. 2021;5(1). https://doi.org/10.1038/s41529-021-00151-y

[39] Saha S, Gan Z, Cheng L, Gao J, Kafka OL, Xie X, et al. Hierarchical Deep Learning Neural Network (HiDeNN): An artificial intelligence (AI) framework for computational science and engineering. Computer Methods in Applied Mechanics and Engineering. 2021;373:113452. https://doi.org/10.1016/j.cma.2020.113452

[40] Zhang L, Cheng L, Li H, Gao J, Yu C, Domel R, et al. Hierarchical deep-learning neural networks: finite elements and beyond. Computational Mechanics. 2021;67:207-30. https://doi.org/10.1007/s00466-020-01928-9

[41] Wang Y, Soutis C, Ando D, Sutou Y, Narita F. Application of deep neural network learning in composites design. European Journal of Materials. 2022;2(1):117-70. https://doi.org/10.1080/26889277.2022.2053302

[42] Saxena A. An Introduction to Convolutional Neural Networks. International Journal for Research in Applied Science & Engineering Technology 2022;10:943-947. https://doi.org/10.22214/ijraset.2022.47789

[43] Indolia S, Goswami AK, Mishra SP, Asopa P. Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach. Procedia Computer Science 2018;132:679-88. https://doi.org/10.1016/j.procs.2018.05.069

[44] Abueidda DW, Almasri M, Ammourah R, Ravaioli U, Jasiuk IM, Sobh NA. Prediction and optimization of mechanical properties of composites using convolutional neural networks. Composite Structures 2019;227:111264. https://doi.org/10.1016/j.compstruct.2019.111264

[45] Cang R, Li H, Yao H, Jiao Y, Ren Y. Improving direct physical properties prediction of heterogeneous materials from imaging data via convolutional neural network and a morphology-aware generative model. Computational Materials Science. 2017;150:212. https://doi.org/10.1016/j.commatsci.2018.03.074

[46] Li X, Liu Z, Cui S, Luo C, Li C, Zhuang Z. Predicting the effective mechanical property of heterogeneous materials by image based modeling and deep learning. Computer Methods in Applied Mechanics and Engineering. 2019;347:735-53. https://doi.org/10.1016/j.cma.2019.01.005

[47] Rao C, Liu Y. Three-dimensional convolutional neural network (3D-CNN) for heterogeneous material homogenization. Computational Materials Science. 2020;184:109850. https://doi.org/10.1016/j.commatsci.2020.109850

[48] Kelleher JD. Deep learning. MIT Press. The MIT Press, Massachusetts Institute of Technology; 2021

[49] Lipton ZC, Berkowitz J, Elkan C. A Critical Review of Recurrent Neural Networks for Sequence Learning. arXiv.org. 2015;1506.00019

[50] Pascanu R, Gulcehre C, Cho K, Bengio Y. How to construct deep recurrent neural networks. Proceedings of the Second International Conference on Learning Representations (ICLR 2014) 2014

[51] Hochreiter S, Schmidhuber J. Long Short-Term Memory. Neural Computation. 1997;9(1):1-42. https://doi.org/10.1162/neco.1997.9.1.1

[52] Cho K, van Merrienboer B, Bahdanau D, Bengio Y. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. arXiv.org. 2014;1409.1259

[53] Mozaffar M, Bostanabad R, Chen W, Ehmann K, Cao J, Bessa MA. Deep learning predicts path-dependent plasticity. Proceedings of the National Academy of Sciences. 2019;116(52):26414-20. https://doi.org/10.1073/pnas.1911815116

[54] Wu L, Nguyen VD, Kilingar NG, Noels L. A recurrent neural network-accelerated multi-scale model for elasto-plastic heterogeneous materials subjected to random cyclic and non-proportional loading paths. Comput Methods Appl Mech Eng. 2020 ;369(113234):113234. https://doi.org/10.1016/j.cma.2020.113234

[55] Logarzo HJ, Capuano G, Rimoli JJ. Smart constitutive laws: Inelastic homogenization through machine learning. Comput Methods Appl Mech Eng. 2021;373(113482):113482. https://doi.org/10.1016/j.cma.2020.113482

[56] Bonatti C, Mohr D. On the importance of self-consistency in recurrent neural network models representing elasto-plastic solids. Journal of the Mechanics and Physics of Solids 2022;158:104697. https://doi.org/10.1016/j.jmps.2021.104697

[57] Ghavamian F, Simone A. Accelerating multiscale finite element simulations of history-dependent materials using a recurrent neural network. Comput Methods Appl Mech Eng 2019;357(112594):112594. https://doi.org/10.1016/j.cma.2019.112594

[58] Schwarzer M, Rogan B, Ruan Y, Song Z, Lee DY, Percus AG, et al. Learning to fail: Predicting fracture evolution in brittle material models using recurrent graph convolutional neural networks. Computational Materials Science. 2019;162:322-32. https://doi.org/10.1016/j.commatsci.2019.02.046

[59] Koeppe A, Bamer F, Hernandez Padilla CA, Markert B. Neural network representation of a phase-field model for brittle fracture. PAMM. 2017;17(1):253-4. https://doi.org/10.1002/pamm.201710096

[60] Karniadakis GE, Kevrekidis IG, Lu L, Perdikaris P, Wang S, Yang L. Physics-informed machine learning. Nature Reviews Physics. 2021;3(6):422-40. https://doi.org/10.1038/s42254-021-00314-5

[61] Das S, Tesfamariam S. State-of-the-Art Review of Design of Experiments for Physics-Informed Deep Learning. arxiv.org 2022;2202.06416

[62] Güneş Baydin A, Pearlmutter B, Siskind J, Baydin G, Radul A, Mark J. Automatic Differentiation in Machine Learning: a Survey. Journal of Machine Learning Research. 2018;18:1-43.

[63] Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics. 2019;378:686-707. https://doi.org/10.1016/j.jcp.2018.10.045

[64] Haghighat E, Raissi M, Moure A, Gomez H, Juanes R. A deep learning framework for solution and discovery in solid mechanics. arXiv.org 2020;2003.02751

[65] Arora R, Kakkar P, Dey B, Chakraborty A. Physics-informed neural networks for modeling rate- and temperature-dependent plasticity. arXiv.org 2022;2201.08363

[66] Bai J, Rabczuk T, Gupta A, Alzubaidi L, Gu Y. A physics-informed neural network technique based on a modified loss function for computational 2D and 3D solid mechanics. Comput Mech. 2023;71(3):543-62. https://doi.org/10.1007/s00466-022-02252-0

[67] Abueidda DW, Lu Q, Koric S. Meshless physics-informed deep learning method for three-dimensional solid mechanics. International Journal for Numerical Methods in Engineering. 2021;7182-7201. https://doi.org/10.1002/nme.6828

[68] He Z, Ni F, Wang W, Zhang J. A physics-informed deep learning method for solving direct and inverse heat conduction problems of materials. Materials Today Communications 2021;2:102719. https://doi.org/10.1016/j.mtcomm.2021.102719

[69] Cai S, Wang Z, Wang S, Perdikaris P, Karniadakis G. Physics-Informed Neural Networks (PINNs) for Heat Transfer Problems. Journal of Heat Transfer 2021;143. https://doi.org/10.1115/1.4050542

[70] Li W, Bazant MZ, Zhu J. A physics-guided neural network framework for elastic plates: Comparison of governing equations-based and energy-based approaches. Computer Methods in Applied Mechanics and Engineering. 2021;383:113933. https://doi.org/10.1016/j.cma.2021.113933

[71] Goswami S, Anitescu C, Chakraborty S, Rabczuk T. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. Theoretical and Applied Fracture Mechanics 2020;106:102447. https://doi.org/10.1016/j.tafmec.2019.102447

[72] Zhou J, Gao J, Wang K, Liao Y. Design Optimization of a Disc Brake Based on a Multi-Objective Optimization Algorithm and Analytic Hierarchy Process Method. Transactions of FAMENA. 2019;42(4):25-42. https://doi.org/10.21278/TOF.42403

[73] Bischof C, Wilfinger D. Big Data-Enhanced Risk Management. Transactions of FAMENA. 2019;43(2):73-84. https://doi.org/10.21278/TOF.43206

[74] Vijayan S, Parameshwaran Pillai T. Application of a Machine Learning Algorithm in a Multi Stage Production System. Transactions of FAMENA. 2022;46(1):91-102. https://doi.org/10.21278/TOF.461033121

[75] ABAQUS. ABAQUS/Standard in: ABAQUS, Dassault Systemes. Providence. USA. 2014

[76] Lesičar T, Tonković Z, Sorić J. A Second-Order Two-Scale Homogenization Procedure Using C1 Macrolevel Discretization. Computational mechanics. 2014;54(2):425-41. https://doi.org/10.1007/s00466-014-0995-3

[77] Abadi M, Agarwal A, Barham P, et al. TensorFlow: Large-scale machine learning on heterogeneous systems 2015. URL: https://www.tensorflow.org/

[78] Cholett F, et al. Keras. 2015. URL: https://keras.io/

[79] Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. arXiv.org. 2014;1412.6980

[80] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. JMLR Workshop and Conference Proceedings 2010; 249-56

[81] Olah C. Understanding LSTM Networks. LSTMblog. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[82] Simo JC, Hughes TJR. Computational Inelasticity. New York: Springer-Verlag; 1998.

[83] Sorić J. Finite Element Method, Linear and Nonlinear Analysis of Structures. Golden marketing-Tehnička knjiga. Zagreb 2021 (in Croatian)

[84] Werbos PJ. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE. 1990;78(10):1550-60. https://doi.org/10.1109/5.58337

Submitted: 14.4.2023

Accepted: 24.5.2023

Jurica Sorić, Professor Emeritus
Matej Stanić, Mag. Ing. Mech.
Tomislav Lesičar*, Assistant Professor
Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Zagreb, Croatia
*Corresponding author:
tomislav.lesicar@fsb.hr