Efficient implementation of non-linear flow law using neural network into the Abaqus Explicit FEM code

Olivier Pantalé, Pierre Tize Mha, Amèvi Tongne

Efficient implementation of non-linear flow law using neural network into the Abaqus Explicit FEM code

Olivier Pantalé *, Pierre Tize Mha, Amèvi Tongne

Laboratoire Génie de Production, INP/ENIT, Université de Toulouse, 47 Av d'Azereix, Tarbes, 65016, France

Keywords:
Artificial Neural Network
Constitutive behavior
Finite element method
Numerical implementation
Johnson–Cook flow law

A B S T R A C T

Machine learning techniques are increasingly used to predict material behavior in scientific applications and offer a significant advantage over conventional numerical methods. In this work, an Artificial Neural Network (ANN) model is used in a finite element formulation to define the flow law of a metallic material as a function of plastic strain ε^p, plastic strain rate $\dot{\varepsilon}^p$ and temperature T. First, we present the general structure of the neural network, its operation and focus on the ability of the network to deduce, without prior learning, the derivatives of the flow law with respect to the model inputs. In order to validate the robustness and accuracy of the proposed model, we compare and analyze the performance of several network architectures with respect to the analytical formulation of a Johnson–Cook behavior law for a 42CrMo4 steel. In a second part, after having selected an Artificial Neural Network architecture with 2 hidden layers, we present the implementation of this model in the Abaqus Explicit computational code in the form of a VUHARD subroutine. The predictive capability of the proposed model is then demonstrated during the numerical simulation of two test cases: the necking of a circular bar and a Taylor impact test. The results obtained show a very high capability of the ANN to replace the analytical formulation of a Johnson–Cook behavior law in a finite element code, while remaining competitive in terms of numerical simulation time compared to a classical approach.

1. Introduction

Numerical simulation of forming processes, machining or the behavior of structures subjected to dynamic loads and impacts requires the use of specific material behavior laws, whose parameters are identified by tests based on Taylor impacts, Hopkinson bars or Gleeble thermomechanical simulator. The behavior laws are selected according to their availability in a finite element code or the possibility, if not available, to implement them through user subroutines. In this study, our work is based on the use of the finite element code Abaqus Explicit which offers the possibility to define user behavior laws through FORTRAN subroutines VUMAT or VUHARD [1,2] like the work proposed by Duc-Toan et al. [3] or more recently by Ming et al. [4]. The usual procedure is to select a mathematical form of the behavior law among those available in the literature (Johnson–Cook, Zerilli Armstrong, ...) and then, from the results of experimental tests, to identify via a regression method, the parameters of the selected law.

In most cases, the behavior of the material at high temperatures and strain rates is highly nonlinear, and the effects of many factors on the flow stress are also nonlinear, which reduces the accuracy of the prediction by the regression methods usually used and limits the field of application. In addition, the selection, development, and numerical implementation of such constitutive equations is time-consuming. Artificial intelligence techniques allow advances concerning the laws of behavior in order to allow a better identification of these laws. Thus, Versino et al. [5] used a Machine Learning technique based on symbolic regression for the development of data-driven constitutive model. Obtaining a flow equation, and thus its analytical derivative, allows the use of iterative solvers that employ Jacobians (i.e., the Newton–Raphson scheme) that allow a higher order of convergence. This symbolic regression technique has also been used by other authors since, such as Bomarito et al. [6], Park et al. [7] using constrained symbolic regression technique, or Nassr et al. [8] using evolutionary polynomial regression.

Given this situation, it is therefore natural to look for a method to eliminate some intermediate steps between experimental tests and numerical simulation in order to simplify the computational chain. In this perspective, recent advances in deep learning constitute a way of investigation. The basic idea is to replace the analytical formulation used to calculate the flow stress σ of the material as a function of the plastic strain ε^p, the plastic strain rate $\dot{\varepsilon}^p$ and the temperature T, by

* Corresponding author.
E-mail address: Olivier.Pantale@enit.fr (O. Pantalé).
URL: http://www.enit.fr (O. Pantalé).

an Artificial Neural Network (ANN). This neural network is trained to reproduce the behavior of the considered material only from the experimental data resulting from the tests, ignoring any assumption on the analytical form of the assumed flow law. Consequently, it is no longer necessary to postulate an analytical form of the behavior law in order to implement it in a FEM code.

Artificial neural networks and deep learning are becoming more and more important in today's society, and their fields of application are getting wider and wider. After a boom in the early 1990s and a decline in interest towards the end of the 20th century, neural networks are experiencing a resurgence of interest and even a huge media hype under the name of deep learning. Their use in science and physics is now widespread, notably because of the current availability of efficient tools allowing to program Artificial Neural Networks thanks to widely available libraries such as Tensorflow [9] for example. The most publicized applications of deep learning are mainly related to medical diagnosis, robotics, images and language recognition, but the applications go far beyond that and all sciences can now use these techniques, including thermomechanical numerical simulation.

Artificial neural networks can solve problems that are difficult to conceptualize using traditional computational methods. Unlike a classical approach based on a regression method, an Artificial Neural Network does not need to know the mathematical form of the model it seeks to reproduce. The Artificial Neural Network learns from the training data and can reproduce the behavior of a model from the simple knowledge of a series of input and output values with no prior assumption on their nature and their interrelations. Hornik et al. [10] have rigorously established in 1989 that feed-forward Neural Network are a class of universal approximators, extending the work proposed twenty years before by Minsky and Papert [11] where they demonstrated that the simple two-layer perceptron is incapable of usefully representing or approximating functions outside a very narrow and special class. The ANN has adjustment, memorization and anticipation capabilities, and better performances than the approach based on implementing a constitutive equation. Therefore, Artificial Neural Networks can now enable novel approaches for modeling the behavior of materials and have been successfully applied in the prediction of constitutive relationships of some metals and alloys in recent years. Applicability of ANN to model path dependent plasticity has been explored and some review of the literature can be found for example in Gorji et al. [12] concerning the use of Recurrent Neural Network, in Jamli et al. [13] concerning their application in finite element analysis of metal forming processes, or in Jiao et al. [14] concerning the applicability to meta-materials and their characterization. A distinction must be made between ANN-based hardening models and ANN-based constitutive models. Both approaches have been studied by many researchers over the last thirty years. Ghaboussi et al. [15] published a pioneering paper, in which they proposed an ANN-based constitutive model for planar concrete under monotonic biaxial loading and cyclic uniaxial loading in which they successfully predicted several loading paths in the biaxial loading condition. They improved NN architecture by introducing Adaptive NN and Autoprogressive NN in [16,17] where the network architecture evolves during the training phase to better learn complex stress–strain behavior of materials using a global load–deflection response. The approach adopted for this study is an ANN-based hardening model for which, the evaluation of the material flow stress calculated by the ANN is combined with a Radial Return type integration scheme.

Lin et al. [18] developed a neural network to predict the flow stress of 42CrMo4 steel in hot compression tests on a Gleeble thermomechanical device and showed a very good correlation between experimental and predicted results. An extension to the numerical implementation of this approach would have been desirable. Javadi et al. [19] used a neural network to capture the behavior of complex materials using a FE model incorporating a backpropagation neural network. Lu et al. [20] presented a comparative study of the modeling of an Al–Cu–Mg–Ag alloy behavior by a constitutive equation based on the Zener–Hollomon parameter and a neural network. It also shows that the model based on the ANN proposes a better prediction than the constitutive equation. Ashtiani et al. [21] compared the prediction capabilities of an ANN against a conventional approach based on several behavior laws such as Johnson–Cook, Arrhenius and Strain compensated Arrhenius and concluded better efficiency and accuracy of the neural network in predicting the hot behavior of Al–Cu–Mg–Pb alloy. Ashtiani et al. [21] have shown that a well-trained ANN can efficiently overcome the lacks of physics coming from analytical constitutive behaviors such as the Johnson–Cook, or the Arrhenius one. Ali et al. [22] presented an ANN model coupled with a rate dependent crystal plasticity finite element method to simulate the stress–strain behavior of material and its microstructure evolution in a AA6063-T6 under simple shear and tension. Stoffel et al. [23,24] applied ANN to complicated structural deformations of shock-wave loaded plates involving both geometrical and physical non-linearities. Li et al. [25] implemented a VUMAT subroutine for Abaqus where parameters were identified through a combination of analytical formulas and a back propagation algorithm. Recently, Huang et al. [26] developed a neural network model to predict the flow stress and the microstructure evolution of Ti–6Al–4V alloy. It also showed the superiority of this approach over an Arrhenius behavior model, especially because the ANN can predict the flow stress in the whole range of deformation. Temporal Convolutional Networks have also been applied by Abueidda et al. [27] to predict the history-dependent responses of a class of cellular materials. Thermodynamics based ANN where also proposed by Masi et al. [28] to reduce physically inconsistencies in the predictions of the NN. They demonstrate the wide applicability of TANNs for modeling elasto-plastic materials, using both hyper- and hypo-plasticity models. As presented by Knight et al. [29], evolution of Neural Network Architectures is not the only way to progress, constant evolution of the hardware architecture to implement ANN will also have to be taken into account for the next future.

In Section 2, we present the main bases of the deep learning with in details the description of the structure of the neural network and the equations which govern its functioning. As we will see in Section 4 concerning the numerical implementation of the flow law, the programming of the neural network on the finite element code Abaqus Explicit requires the determination of the 3 derivatives of the flow stress σ with respect to the plastic strain ε^p, the plastic strain rate $\dot{\varepsilon}^p$ and the temperature T. The determination of these derivatives will be the subject of the second part of Section 2. Section 3 is devoted to a detailed presentation of the ANN learning for a Johnson–Cook type behavior law for a 42CrMo4 steel. In this section, we will show the influence of the network structure on the accuracy of the flow stress and the derivatives evaluation. Section 4 is devoted to a presentation of the numerical implementation of the neural network in the Abaqus finite element code in the form of a FORTRAN VUHARD subroutine as well as numerical test cases to validate the proposed approach. A conclusion and perspectives are finally proposed at the end.

2. Artificial neural network set-up

In this section, we briefly introduce the basic concepts of backward and forward propagating Artificial Neural Networks (ANNs) that are relevant to this work. The global architecture we have retained for this work is a multi-layer feed-forward network which can be seen as a universal approximator as proposed by Hornik et al. [10]. The proposed neural network is used to approximate non-linear functions. The concept of neural network consists in simulating the flow of information inside the human brain by defining a set of neurons (arranged in different layers) defining functions producing results according to the inputs. These neurons are interconnected from one layer to another and continuously improve their predictive ability as the network is trained and learns a new concept [30]. Fig. 1 presents the global architecture of an ANN with multiple hidden layers where neurons are set up in different layers (from the first hidden layer to the output layer) with the following characteristics:

- The first layer, is the so-called input layer, and in our application concerning constitutive law, consists of 3 inputs. This one is not composed of neurons as they will be defined hereafter, and collects the incoming information: the three values corresponding to the plastic strain ε^p, the plastic strain rate $\dot{\varepsilon}^p$ and the temperature T respectively.
- The ANN contains at least one hidden layer (in the current paper, we will consider only one and two hidden layers ANNs) containing a variable number of neurons.
- The last layer is the so-called output layer, and in our case consists of only one neuron providing the value of the von Mises flow stress σ.
- Neurons are not connected to other neurons of the same layer but only to the immediately preceding outputs and following inputs.

In Fig. 1 we have separated the summation $\vec{y}^{(k)}$ and the activation $\vec{\hat{y}}^{(k)}$ parts of all neurons within the hidden layers. Concerning the notation used, note here that we are using the formalism of a vector (using an arrow over the symbol) for many quantities in the equations hereafter, even if these are not vectors in the sense of a mathematical notation. In reality, these quantities are vertical one-dimensional arrays used to store data, which visually correspond to the components of a vector. In the proposed architecture, there is no activation function associated to the neuron in the output layer, as usually done for regression problems which is our case here. The data flows from layer to layer until we obtain the output of the ANN. Every layer, except the first one, in the ANN, no matter how many neurons n are in, have an input with a variable number of items m and an output with a fixed number of items n. In the so-called forward propagation algorithm, the output of one layer becomes the input of the next one.

2.1. Neural network governing equations

2.1.1. Input layer

Conforming to Fig. 1, the input layer is defined by $\vec{x} = [x_1, x_2, x_3]^T$. In the proposed application, $\vec{x}$ has three components x_i related to the plastic strain ε^p, the plastic strain rate $\dot{\varepsilon}^p$ and the temperature T, respectively. As it will be presented in Section 2.1.4, input variables x_i are normalized to avoid ill-conditioning for the optimization procedure, mainly because they represent various physics with a large discrepancy in values.

2.1.2. Hidden layers

Any hidden layer k, containing n neurons, takes a weighted sum of the outputs $\vec{x}$ of the immediately previous layer $(k-1)$, containing m neurons, given by the following equation:

$$y_i^{(k)} = \sum_{j=1}^{m} w_{ij}^{(k)} x_j + b_i^{(k)} \tag{1}$$

where $y_i^{(k)}$ is the nodal value of the ith neuron of layer k, $w_{ij}^{(k)}$ is the associated weight parameter between the ith neuron of layer k and the jth neuron of layer $(k-1)$ and $b_i^{(k)}$ is the associated bias of the ith neuron of layer k. Those weights and bias are the training parameters of the ANN that we have to adjust during the training procedure described in Section 3. Using matrix notation, Eq. (1) can be rewritten with the following form:

$$\vec{y}^{(k)} = \mathbf{W}^{(k)} \cdot \vec{x} + \vec{b}^{(k)} \tag{2}$$

where $\vec{y}^{(k)} = [y_1^{(k)}, y_2^{(k)}, \dots, y_n^{(k)}]^T$ contains the nodal values resulting from the summation operation in layer k, $\vec{b}^{(k)} = [b_1^{(k)}, b_2^{(k)}, \dots, b_n^{(k)}]^T$ is the nodal bias of layer k and $\mathbf{W}^{(k)}$ is the $[n \times m]$ weight parameters matrix of layer k given hereafter:

$$\mathbf{W}^{(k)} = \begin{bmatrix} w_{11}^{(k)} & w_{12}^{(k)} & \cdots & w_{1m}^{(k)} \\ w_{21}^{(k)} & w_{22}^{(k)} & \cdots & w_{2m}^{(k)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1}^{(k)} & w_{n2}^{(k)} & \cdots & w_{nm}^{(k)} \end{bmatrix} \tag{3}$$

The total number of training parameters N for any hidden layer k is the sum of the number of weight parameters and the number of bias parameters of layer k, so $N = n(m+1)$. After the summation operation defined by Eq. (2), each neuron in the hidden layer k provides an output value $\vec{\hat{y}}^{(k)}$ computed from an activation function $f^{(k)}$ according to the following equation:

$$\hat{y}_i^{(k)} = f^{(k)}\left(y_i^{(k)}\right) \qquad \text{or} \qquad \vec{\hat{y}}^{(k)} = f^{(k)}\left(\vec{y}^{(k)}\right) \tag{4}$$

Many activation functions are available in literature and their choice depend mainly on the application of the ANN. In our case, it is very important that those activation functions are derivable in order to allow the computation of the derivative of the von Mises stress σ with regard to the plastic strain ε^p, the plastic strain rate $\dot{\varepsilon}^p$ and the temperature T. For our type of application, we have made the choice to test two among the mainly used ones:

- the Sigmoid $\mathrm{sig}(x)$ activation function defined by:

$$\mathrm{sig}(x) = \frac{1}{1+\exp(-x)}, \quad \mathrm{sig}'(x) = \frac{\exp(x)}{(1+\exp(x))^2} \tag{5}$$

- and the Hyperbolic tangent $\tanh(x)$ activation function defined by:

$$\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}, \quad \tanh'(x) = 1-\tanh^2(x) \tag{6}$$

The main guide for this choice is that the expression of their derivatives is simple, which leads to relatively compact expressions. Comparison and performance of those two activation functions will be presented in Section 3.

If there exists another hidden layer $(k+1)$ after the current layer, then, the output $\vec{\hat{y}}$ of layer k is then used as the input $\vec{x}$ of layer $(k+1)$.

2.1.3. Output layer

The output s is computed from the values of the last hidden layer l of the neural network, containing m neurons, using the following equation:

$$s = \sum_{j=1}^{m} w_j \hat{y}_j^{(l)} + b \tag{7}$$

where b is the bias associated to the output neuron and w_i are the m weight parameters between the last hidden layer and the output neuron s. Using matrix formalism, one can rewrite this later as:

$$s = \vec{w}^T \cdot \vec{\hat{y}}^{(l)} + b \tag{8}$$

with $\vec{w} = [w_1, w_2, \dots, w_m]^T$. As presented earlier, there is no activation function for the output neuron, so s is directly the output of the neural network. The total number of training parameters for the output layer is $m+1$. For a neural network with two hidden layers having m neurons on the first hidden layer and n neurons on the second one, the total number of training parameters is $N = 4m + n(m+2) + 1$.

2.1.4. Pre and postprocessing of data

Since the ANN are set up to treat values with a limited amplitude, it is necessary to pre-process the provided corresponding values of the plastic strain ε^p, the plastic strain rate $\dot{\varepsilon}^p$ and the temperature T in the range $[0, 1]$ in a same manner as the one proposed by other authors [18, 20]. Therefore, the input $\vec{x}$ of the ANN is computed according to the constitutive flow law by:

- Since the plastic deformation rate in constitutive equations usually enhances the logarithm of the plastic strain rate, we preprocess the plastic strain rate by computing the natural logarithm of the ratio of the plastic strain rate $\dot{\varepsilon}^p$ over the referential strain rate $\dot{\varepsilon_0}$ given by $\ln(\dot{\varepsilon}^p/\dot{\varepsilon_0})$.
- Then, we normalize the x_i variables in the range $[0, 1]$ to avoid ill-conditioned system as presented by many other authors in the literature [18,20].

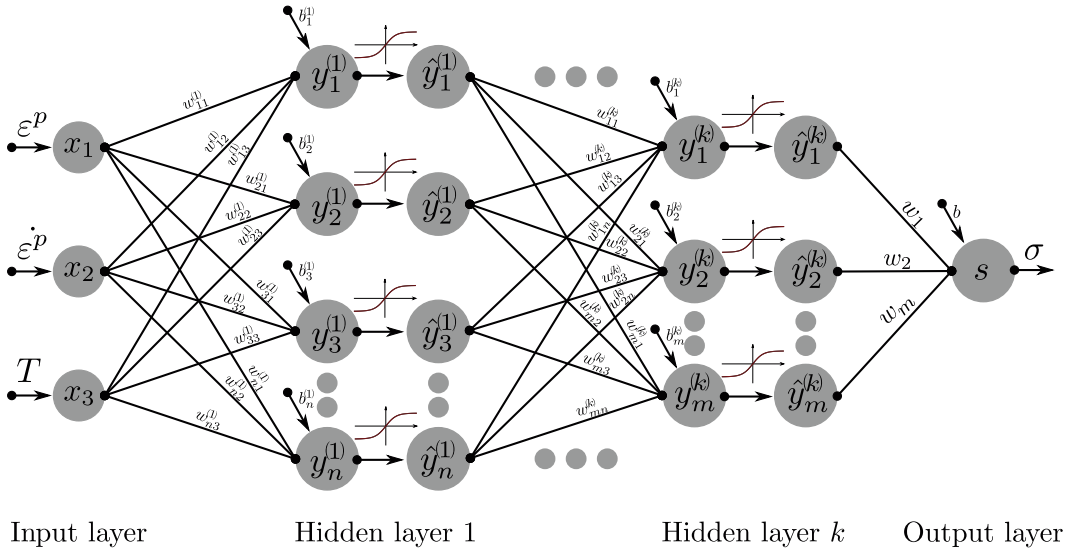

Fig. 1. Multi-layer Artificial Neural Network architecture.

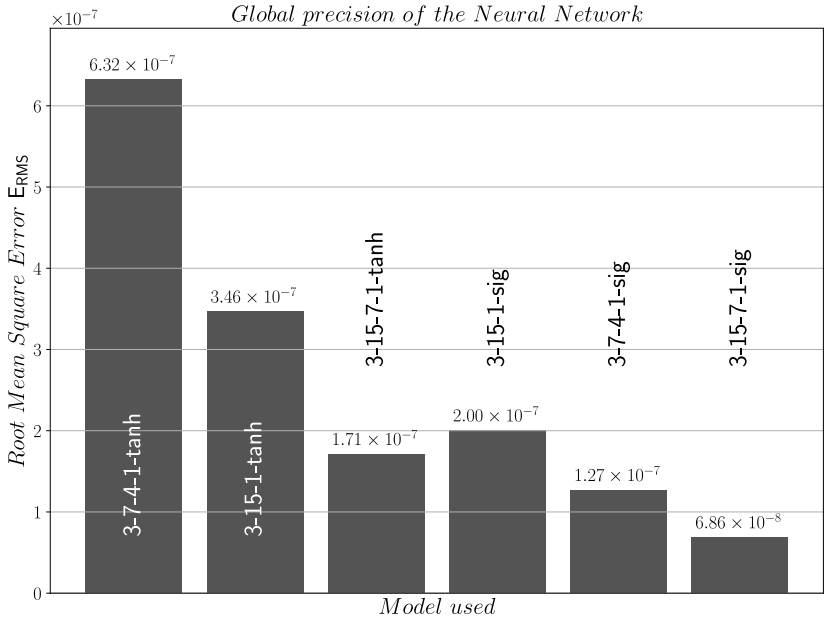

Fig. 2. Comparison of precision of various ANN.

Therefore, the three components of the input $\vec{x}$ are obtained from the plastic strain ε^p, the plastic strain rate $\dot{\varepsilon}^p$ and the temperature T using the following expressions:

$$\vec{x} = \begin{cases} x_1 = \frac{\varepsilon^p - [\varepsilon^p]_{min}}{[\varepsilon^p]_{max} - [\varepsilon^p]_{min}} \\ x_2 = \frac{\ln(\dot{\varepsilon}^p/\dot{\varepsilon}_0) - [\ln(\dot{\varepsilon}^p/\dot{\varepsilon}_0)]_{min}}{[\ln(\dot{\varepsilon}^p/\dot{\varepsilon}_0)]_{max} - [\ln(\dot{\varepsilon}^p/\dot{\varepsilon}_0)]_{min}} \\ x_3 = \frac{T - [T]_{min}}{[T]_{max} - [T]_{min}} \end{cases} \tag{9}$$

where $[\ ]_{min}$ and $[\ ]_{max}$ are the boundaries of the range of the corresponding field. During the training of the ANN, the von Mises stresses σ will also be scaled down within the range $[0, 1]$, using the following expression:

$$s = \frac{\sigma - [\sigma]_{min}}{[\sigma]_{max} - [\sigma]_{min}} \tag{10}$$

So, finally, the von Mises stress σ can be obtained from the output s of the ANN using:

$$\sigma = \left([\sigma]_{max} - [\sigma]_{min}\right) s + [\sigma]_{min} \tag{11}$$

The $[\ ]_{min}$ and $[\ ]_{max}$ values of plastic strain, plastic strain rate, temperature, stresses and referential strain rate $\dot{\epsilon_0}$ used during the training phase should be recorded for later use during the implementation of the ANN in the Abaqus Explicit code. They are part of the final solution along with the weights $\mathbf{W}^{(k)}$ and bias values $\vec{b}^{(k)}$ of the hidden layers and the weights $\vec{w}$ and the bias b of the output layer of the neural network that constitutes the training parameters of the neural network. The knowledge of these quantities after the learning phase allows then to extract the whole neural network from its formulation in Python language to a compact version, without the back propagation learning mechanism, for its implementation in FORTRAN in the Abaqus Explicit FEM code.

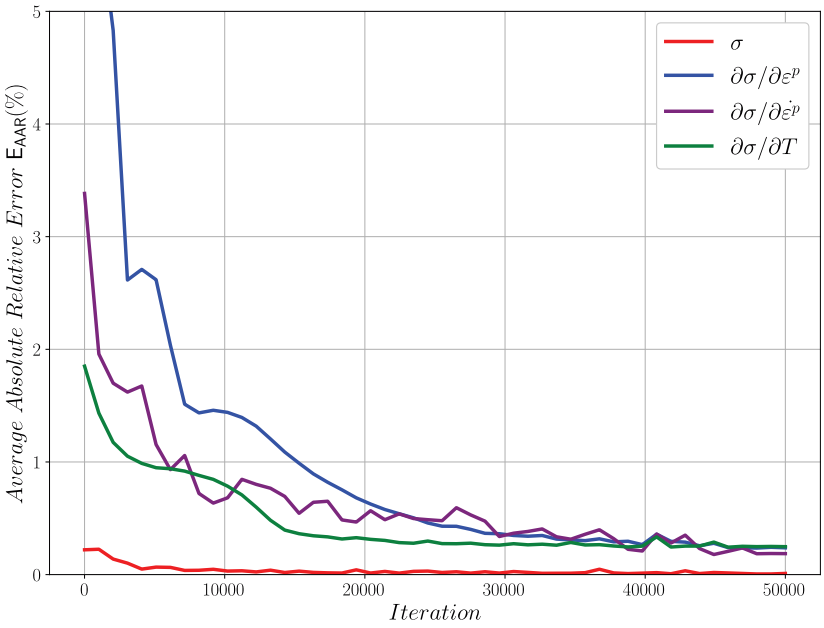

Fig. 3. Convergence of the predicted values for the 3-15-7-1-sig ANN.

2.1.5. Loss function

In the context of optimization algorithms, the function used to evaluate the quality of a solution is called the objective function. In the application to neural networks, we try to minimize the error made by the network in the prediction of the solution. The evaluation of this error consists in measuring the difference between the predicted value σ_i computed by the NN and a reference value σ_i^y coming usually from experiments, or, in the present paper from an analytical equation, that we want to reach for a particular data set. There are several ways to define this error, among them, the best known is probably the average Root Mean Square Error (E_{RMS}) given by:

$$E_{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\sigma_i - \sigma_i^y)^2} \tag{12}$$

where N is the total number of values of the training batch.

2.2. Derivatives computation

Implementation of the neural network within the radial return algorithm proposed by Ming et al. [4] through a VUMAT subroutine of directly through a VUHARD one for Abaqus Explicit requires that the ANN returns not only the von Mises equivalent stress σ given by Eq. (11), but also the three derivatives $\partial\sigma/\partial\varepsilon^p$, $\partial\sigma/\partial\dot{\varepsilon}^p$ and $\partial\sigma/\partial T$ without having been trained to compute those derivatives. So that we do not have 4 outputs for the ANN but only one and the ability to compute those derivatives must be intrinsic. Therefore, we need to be able to compute those three derivatives using only the proposed ANN architecture. One straightforward, but not recommended, solution to this problem is to numerically compute the derivative of σ with respect to ε^p, $\dot{\varepsilon}^p$ and T using the following relation:

$$\frac{\partial\sigma(x)}{\partial x} = \frac{\sigma(x+\delta x)-\sigma(x)}{\delta x} \tag{13}$$

where δx is a small increase ($\delta x = 10^{-6}$ for example) applied to one of the 3 variables ε^p, $\dot{\varepsilon}^p$ and T. We need to compute 4 times a result from the ANN to compute the flow stress and approximate the three derivatives which is quite time-consuming.

Another way to compute those derivatives is to compute the quantity $\vec{s}' = [s_1', s_2', s_3']^T$ containing the 3 derivatives of the output s defined by Eq. (8) of the ANN with respect to the input $\vec{x}$. This analytic computation depends on the number of hidden layers (hereafter we present the results for 1 and 2 hidden layers), and the type of activation functions used (the tanh and the sig functions). $\vec{s}' = \partial s/\partial\vec{x}$ contains the derivatives of s with respect to x_1, x_2 and x_3 respectively. Based on the chain-rule derivation, with 1 hidden layer and a tanh activation function, $\vec{s}'$ takes the following form:

$$\vec{s}' = \mathbf{W}^{(1)^T} \cdot \left[\vec{w} - \vec{w} \circ \tanh^2\left(\vec{y}^{(1)}\right)\right] \tag{14}$$

where $\vec{y}^{(1)}$ is given by Eq. (2) with $k = 1$ and $\circ$ is the element-wise product, known as the Hadamard product, which is a binary operation that takes two matrices A and B of the same dimensions and produces another matrix C of the same dimension as the operands, where each element $C_i = A_i\ B_i$. The tanh operator applied on quantity $\vec{y}^{(1)}$ is just computed for each component of $\vec{y}^{(1)}$, as presented in Fig. 1, since this is not a real but only an array of reals stacked vertically. With 1 hidden layer and a sig activation function, we obtain:

$$\vec{s}' = \mathbf{W}^{(1)^T} \cdot \left[\frac{\vec{w} \circ \exp\left(-\vec{y}^{(1)}\right)}{\left[1+\exp\left(-\vec{y}^{(1)}\right)\right]^2}\right] \tag{15}$$

When the number of hidden layers increases, the complexity of the derivative increases, so, for 2 hidden layers and a tanh activation function for both layers we obtain:

$$\vec{s}' = \mathbf{W}^{(1)^T} \cdot \left[\mathbf{W}^{(2)^T} \cdot \left(\vec{w} - \vec{w} \circ \tanh^2\left(\vec{y}^{(2)}\right)\right) \circ \left(1 - \tanh^2\left(\vec{y}^{(1)}\right)\right)\right] \tag{16}$$

Finally, for 2 hidden layers and a sig activation function for both layers we obtain:

$$\vec{s}' = \mathbf{W}^{(1)^T} \cdot \left[\mathbf{W}^{(2)^T} \cdot \left(\frac{\vec{w} \circ \exp\left(-\vec{y}^{(2)}\right)}{\left[1+\exp\left(-\vec{y}^{(2)}\right)\right]^2}\right) \circ \left(\frac{\exp\left(-\vec{y}^{(1)}\right)}{\left[1+\exp\left(-\vec{y}^{(1)}\right)\right]^2}\right)\right] \tag{17}$$

Depending on the number of hidden layers and the type of activation functions used, from Eqs. (14) to (17) and because of the pre and post-processing of the quantities σ, ε^p, $\dot{\varepsilon}^p$ and T defined in Section 2.1.4 by Eqs. one can finally obtain the derivatives of the flow stress σ with respect to ε^p, $\dot{\varepsilon}^p$ and T using the following expression:

$$\begin{cases} \partial\sigma/\partial\varepsilon^p = s_1' \frac{[\sigma]_{max}-[\sigma]_{min}}{[\varepsilon^p]_{max}-[\varepsilon^p]_{min}} \\ \partial\sigma/\partial\dot{\varepsilon}^p = \frac{s_2'}{\dot{\varepsilon}^p} \frac{[\sigma]_{max}-[\sigma]_{min}}{[\dot{\varepsilon}^p]_{max}-[\dot{\varepsilon}^p]_{min}} \\ \partial\sigma/\partial T = s_3' \frac{[\sigma]_{max}-[\sigma]_{min}}{[T]_{max}-[T]_{min}} \end{cases} \tag{18}$$

It is important to note here, that whatever the method adopted to calculate the derivative of the output of the neural network with respect to the inputs (numerical method or formulation of the derivative of the network), the result obtained is an approximation of the derivative of the mathematical function represented by the network with respect to the inputs. Indeed, the neural network, by its construction, approximates a mathematical formulation, but, as shown by Nguyen et al. [31], a neural network can also approximate the derivatives of the mathematical function it reproduces. Thus, the derivative of the neural network with respect to its inputs is correlated to the derivative of the mapped mathematical function with respect to its parameters.

3. Training of the ANN and performance evaluation

In order to evaluate the performance of the proposed approach, we decided to reproduce the behavior of the Johnson–Cook [32] flow law with an Artificial Neural Network because it is one of the most widely used flow law for the simulation of high strain rate deformation processes. It is implemented in numerous Finite Element codes such as Abaqus. Reproducing the behavior of a Johnson–Cook law by a neural network is of course not an aim of this work, but only required in order to verify numerically that the neural network is able to take into account a non-linear behavior and to have a way to measure exactly the prediction errors of the neural network. The general formulation $\sigma^y(\varepsilon^p, \dot{\varepsilon}^p, T)$ is given by the following equation:

$$\sigma^y = \left(A + B\varepsilon^{p^n}\right)\left[1 + C\ln\left(\frac{\dot{\varepsilon}^p}{\dot{\varepsilon}_0}\right)\right]\left[1 - \left(\frac{T - T_0}{T_m - T_0}\right)^m\right] \tag{19}$$

where $\dot{\varepsilon}_0$ is the reference strain rate, T_0 and T_m are the reference temperature and the melting temperature of the material respectively and A, B, C, n and m are the five constitutive flow law parameters that we usually have to determine using for example an inverse identification procedure as the one proposed by Dalverny et al. [33] from Taylor impact tests.

Analytical expressions of the three derivatives of the Johnson–Cook flow stress σ^y with respect to ε^p, $\dot{\varepsilon}^p$ and T are given by the three following equations:

$$\begin{cases} \partial\sigma^y/\partial\varepsilon^p &= nB\varepsilon^{p^{n-1}}\left[1 + C\ln\left(\frac{\dot{\varepsilon}^p}{\dot{\varepsilon}_0}\right)\right]\left[1 - \left(\frac{T-T_0}{T_m-T_0}\right)^m\right] \\ \partial\sigma^y/\partial\dot{\varepsilon}^p &= \frac{C}{\dot{\varepsilon}^p}\left(A + B\varepsilon^{p^n}\right)\left[1 - \left(\frac{T-T_0}{T_m-T_0}\right)^m\right] \\ \partial\sigma^y/\partial T &= \frac{-m\left(A+B\varepsilon^{p^n}\right)}{T-T_0}\left[1 + C\ln\left(\frac{\dot{\varepsilon}^p}{\dot{\varepsilon}_0}\right)\right]\left(\frac{T-T_0}{T_m-T_0}\right)^m \end{cases} \tag{20}$$

The usual approach to implement a new constitutive law in the Abaqus FEM code, as proposed by Ming et al. [4], is to program in a VUHARD FORTRAN subroutine the evaluation of the flow stress defined by Eq. (19) and the derivatives of the flow stress given by Eq. (20).

3.1. Generation of the training and test data

A 42CrMo4 steel, with material parameters proposed by Sattouf et al. [34] and reported in Table 1 was selected as the material used in this study. To train and validate the ANN, we have used a Python program to generate, thanks to Eqs. (19)–(20) two distinct data sets:

- The first data set, the training one, contains 2 520 datapoints defined by 70 equidistant values for $\varepsilon^p \in [0, 1]$, 6 plastic strain rates $\dot{\varepsilon}^p \in [1, 10, 50, 500, 5\,000, 50\,000]$ and 6 temperatures $T \in [20, 100, 200, 300, 400, 500]$.
- The second data set, the testing set, contains 5 000 datapoints randomly generated within the ranges $\varepsilon^p \in [0, 1]$, $\dot{\varepsilon}^p \in [1, 50\,000]$ and $T \in [20, 500]$. This later is not used during the training phase, but only during the validation presented in Section 3.3.

Table 1
Material properties of the 42CrMo4 steel [35].

E (GPa)	ν	A (MPa)	B (MPa)	C	n	m
206.9	0.29	806	614	0.0089	0.168	1.1
$\dot{\varepsilon}_0$ (s^{-1})	T_0 °C	T_m °C	ρ kg/m^3	C_p $J/kg°C$	α $10^{-6}/°C$	η
1	20	1540	7830	460	12.3	0.9

Table 2
Global performance analysis of the ANN during the training phase.

Model	N	E_{RMS} $\times 10^{-7}$	$\Delta\sigma$ %	$\Delta(\partial\sigma/\partial\varepsilon^p)$ %	$\Delta(\partial\sigma/\partial\dot{\varepsilon}^p)$ %	$\Delta(\partial\sigma/\partial T)$ %
3-7-4-1-tanh	65	6.32	0.038	1.977	0.792	0.556
3-15-1-tanh	78	3.46	0.039	1.506	0.269	0.371
3-15-7-1-tanh	180	1.71	0.030	0.519	0.380	0.408
3-15-1-sig	78	2.00	0.030	0.686	0.521	0.675
3-7-4-1-sig	65	1.27	0.024	0.670	0.415	0.499
3-15-7-1-sig	180	0.68	0.011	0.247	0.199	0.256

Both data sets contains the values of the plastic strain ε^p, the plastic strain rate $\dot{\varepsilon}^p$, the temperature T, the flow stress σ^y computed from Eq. (19). The second data set contains also the values of the three derivatives computed from Eq. (20). It is important to remember that the neural network proposed must allow to replace the analytical formulation of the behavior law in order to be able to carry out a numerical simulation from experimental data resulting from thermo-mechanical tests. During these tests, we acquire the temperature, the strain, the strain rate and the stresses. But it is impossible to acquire the derivatives of the stresses with respect to the quantities $\varepsilon^p, \dot{\varepsilon}^p, T$. Therefore, in normal use, it is not possible to include the derivatives in the objective function for training the neural network.

3.2. Training of the artificial neural network

Training the Artificial Neural Network is finding the best set of values for all the training parameters $\mathbf{W}^{(k)}$, $\vec{b}^{(k)}$, $\vec{w}$ and b defined in Section 2.1 in order to reduce the error of the ANN in computing the flow stress. The training set is used in this procedure and the ANN has been implemented using the Tensorflow Python library [36]. Training procedure is based on the use of the Adaptive Moment Estimation (ADAM) [37] optimizer. We used the two activation functions presented earlier, the sig and the tanh functions, and one or two hidden layers with a variable number of neurons in the hidden layers. All models are named after their constitution, where 3-x-1-tanh refers to a one hidden layer ANN with a tanh activation function and x neurons in the hidden layer and 3-x-y-1-sig refers to a two hidden layers ANN with a sig activation function in both layers and x neurons in layer 1 and y neurons in layer 2. All models have been trained for the same number of iterations (50 000 iterations). Training times for all models are more or less the same: around 1 hour on a Dell XPS 13 laptop.

3.3. Performance analysis of the artificial neural network

In order to illustrate the efficiency of the ANN, Fig. 2 and Table 2 reports some results obtained after training 6 different models (2 one hidden layer models and 4 two hidden layers models).

A lot more results have been obtained in this study, but only those 6 models are presented here to illustrate the tendencies. Fig. 2 shows an histogram of the average values of the E_{RMS} defined by Eq. (12) evaluated during the last 5% of the training process giving an idea of the global convergence of the ADAM algorithm. In Table 2, N is the number of internal parameters of the ANN. Values $\Delta\square$ are the so-called Average Absolute Relative Error (E_{AAR}) given by the following

expression:

$$\Delta\square = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{\square_i^e - \square_i^p}{\square_i^e}\right| \tag{21}$$

where $\square_i^e$ is the analytical exact value obtained from Eqs. (19)–(20) and $\square_i^p$ is the ANN predicted value of the same quantity computed by the neural network from Eqs. (11) and (18).

Reading the data in Table 2, the overall performance of the proposed neural networks is very good since the error on the evaluation of the flow stress is about 0.01% for the best performing network while it does not exceed 0.04% for all the tested networks. The evaluation of the derivatives is also very good since the error is about 0.2% for the best performing network while it remains below 2.0% for the $\partial\sigma/\partial\varepsilon^p$ term, 0.8% for the $\partial\sigma/\partial\dot{\varepsilon}^p$ term and 0.6% for the $\partial\sigma/\partial T$ term. Without too much surprise, we notice that globally, the error on the evaluation of the flow stress is lower with a factor of 10 than the error on the evaluation of the derivatives. This is easily explained by the fact that the network has been trained to minimize the error on the flow stress only. Nevertheless, without ever having been optimized for computing the derivatives, we can see that the results obtained are also very good.

Fig. 3 shows the convergence of the 3-15-7-1-sig ANN model concerning the evaluation of the flow stress and the 3 derivatives with the number of iterations. From this later, one can see that the convergence of the ANN on the evaluation of σ is faster than convergence of the derivatives. The previously mentioned factor of 10 between the stress and the derivatives is clearly visible on this graph. If the error on the calculation of the constraint does not evolve after 10 000 iterations of the learning algorithm, the errors on the evaluation of the derivatives require a much more significant number of iterations. It is therefore necessary to continue training this type of model when the convergence criterion of the flow constraint has already been satisfied in order to allow the convergence of the derivatives. We can then be confronted with a problem of over-learning, it is therefore necessary to dimension the size of the neural network as accurately as possible with respect to the application.

4. Implementation of the neural network in Abaqus Explicit

In this section, we now present the numerical implementation of the proposed ANN constitutive flow law. Once the ANN network presented in Section 2 has been trained as presented in Section 3, it is time to use it for the flow stress and its three derivatives computing. The implementation is done by programming a VUHARD subroutine for the Abaqus Explicit finite element code similarly to the approach proposed by Jansen van Rensburg et al. [2]. This VUHARD subroutine is used inside of a Radial-Return algorithm, as illustrated in Fig. 4, to compute the flow stress σ^y and its derivative $\frac{d\sigma^y}{d\Gamma}$ used in the expression of the two quantities $\gamma(\Gamma)$ and $\gamma'(\Gamma)$, thanks to the following equation [4]:

$$\begin{aligned}\frac{d\sigma^y}{d\Gamma} &= \frac{\partial\sigma^y}{\partial\varepsilon^p}\frac{d\varepsilon^p}{d\Gamma} + \frac{\partial\sigma^y}{\partial\dot{\varepsilon}^p}\frac{d\dot{\varepsilon}^p}{d\Gamma} + \frac{\partial\sigma^y}{\partial T}\frac{dT}{d\Gamma}\\ &= \sqrt{\frac{2}{3}}\left(\frac{\partial\sigma^y}{\partial\varepsilon^p} + \frac{1}{\Delta t}\frac{\partial\sigma^y}{\partial\dot{\varepsilon}^p} + \frac{\eta\sigma^y}{\rho C_p}\frac{\partial\sigma^y}{\partial T}\right)\end{aligned} \tag{22}$$

where Γ is the consistency parameter used in the Radial-Return algorithm as defined by Simo et al. [38], Δt is the time increment, η is the Taylor–Quinney coefficient defining the amount of plastic work converted into heat energy, C_p is the specific heat coefficient and ρ is the density of the material. So that, the ANN is used to compute the value of the flow stress σ^y and the three derivatives of the flow stress with respect to ε^p, $\dot{\varepsilon}^p$ and T involved in Eq. (22). This is illustrated in Fig. 4 where the yellow block in the center of the flowchart is where the ANN is used. Since this ANN is in the center of a CPU intensive loop involved for all integration points of the FEM model, this has to be optimized to reduce computing times as it will be presented further. More details on the Radial-Return algorithm can be found in Simo et al. [38] and more details concerning the implementation of this

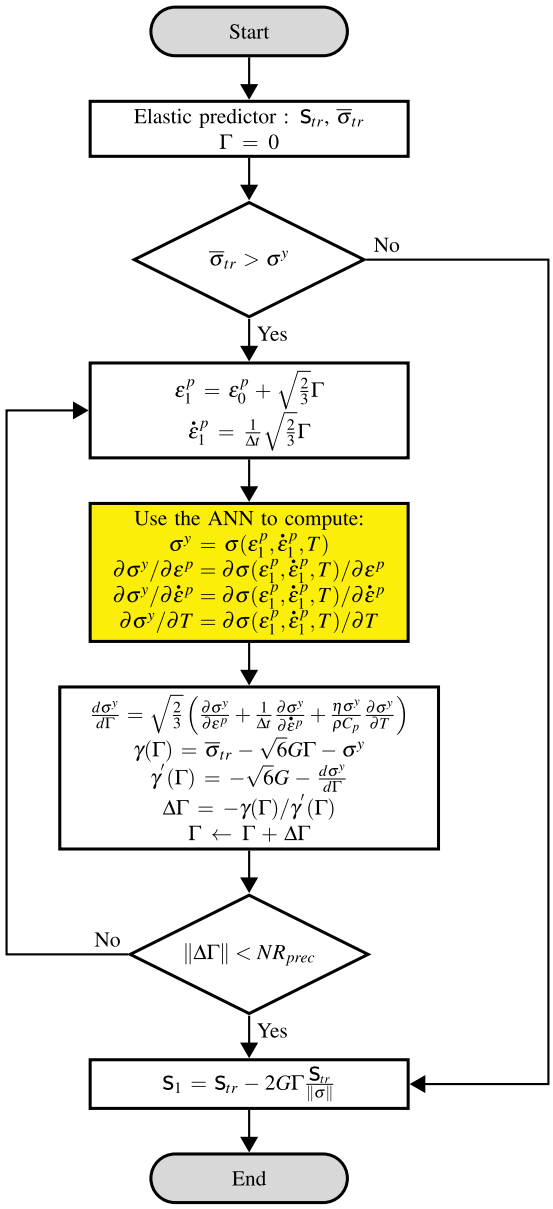

Fig. 4. Flowchart of the Radial-Return Algorithm to compute the final stresses.

algorithm in Abaqus Explicit can be found in Ming et al. [4], where the same approach was used, except the fact that in this paper the flow stress and its derivatives were computed analytically.

Validation of the proposed implementation is done on several benchmark tests by comparing the results obtained using the ANN to native Johnson–Cook law (named Built-in hereafter) and the analytical VUHARD implementations proposed previously by Ming et al. [4] (named Analytical hereafter).

4.1. Numerical implementation of the neural network

The numerical implementation of the neural network is done using a VUHARD subroutine for the Abaqus Explicit code. This is a straightforward approach to implement a new constitutive flow law in this FEM code by just implementing a FORTRAN subroutine to compute the flow stress of the material $\sigma^y(\varepsilon^p, \dot{\varepsilon}^p, T)$ according to Eq. (11) and its derivatives with respect to ε^p, $\dot{\varepsilon}^p$ and T defined by Eq. (18). In this approach, the main part of the Built-In constitutive law is used for time

```
      subroutine vuhard (
     +  nblock, nElement, nIntPt, nLayer, nSecPt, lAnneal, stepTime,
     +  totalTime, dt, cmname, nstatev, nfieldv, nprops, props,
     +  tempOld, tempNew, fieldOld, fieldNew, stateOld, eqps, eqpsRate,
     +  yield, dyieldDtemp, dyieldDeqps, stateNew)
      include 'vaba_param.inc'
      dimension nElement(nblock), props(nprops), tempOld(nblock),
     +  fieldOld(nblock,nfieldv), stateOld(nblock,nstatev),
     +  tempNew(nblock), fieldNew(nblock,nfieldv), eqps(nblock),
     +  eqpsRate(nblock), yield(nblock), dyieldDtemp(nblock),
     +  dyieldDeqps(nblock,2), stateNew(nblock,nstatev)
      character*80 cmname

      do k = 1, nblock
        xepsp = eqps(k)
        xdepsp = log(eqpsRate(k)) / 10.819778
        xtemp = (tempNew(k) - 20.0) / 480.0
        za0 = exp(0.171193*xepsp + 0.498235*xdepsp - 1.572309*xtemp + 0.549710)
        za1 = exp(0.182183*xepsp + 0.279642*xdepsp - 1.381547*xtemp + 1.007667)

        ... rest of the FORTRAN code generated by the Python translator software ...

        Yield(k) = 977.555715*y + 579.184642
        dyieldDeqps(k,1) = 977.555715 * yd0
        dyieldDeqps(k,2) = 90.348959 * yd1 / eqpsRate(k)
        dyieldDtemp(k) = 2.036574 * yd2
      end do
      return
      end
```

Fig. 5. Partial VUHARD FORTRAN subroutine to compute the flow stress and its derivatives using the ANN for the 3-15-7-1-sig model.

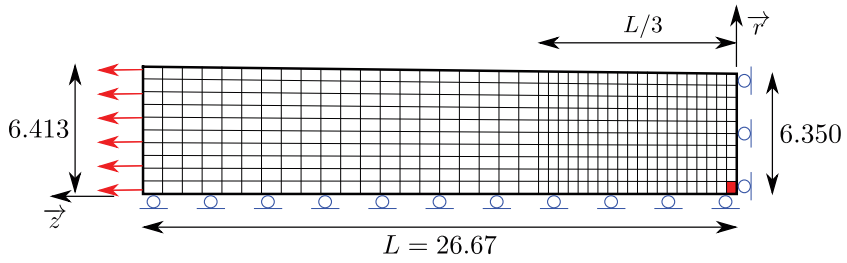

Fig. 6. Numerical model for the necking of a circular bar.

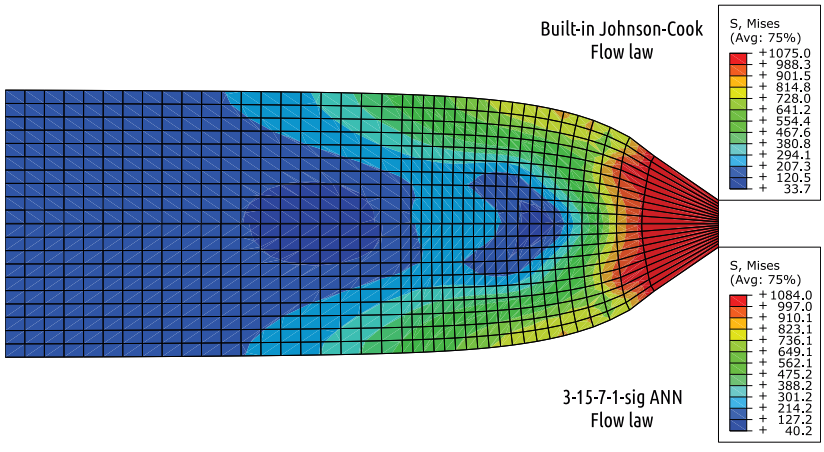

Fig. 7. Von Mises stress $\bar{\sigma}$ contourplot for the necking of a circular bar for an elongation of 7 mm (top side is the Built-in flow law and bottom side is the ANN 3-15-7-1-sig flow law).

integration of the stress, for a given time increment, and the provided user subroutine is called to compute the hardening flow law.

A Python's program has been developed to extract the internal parameters of the trained neural network (the network architecture, the weights and bias values $\mathbf{W}$ and $\vec{b}$ of all layers, ...) and write automatically the FORTRAN subroutine. In the following paragraph, we detail the implementation of a 2 hidden layers neural network with a sig activation function, so that, one of the most complex model presented in this paper.

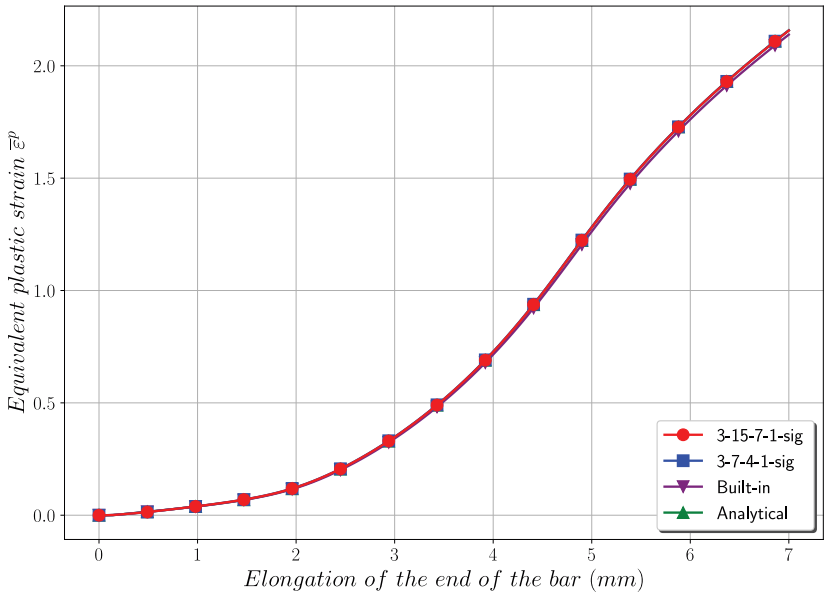

Fig. 8. Equivalent plastic strain $\overline{\varepsilon}^p$ *vs.* displacement for the necking of a circular bar.

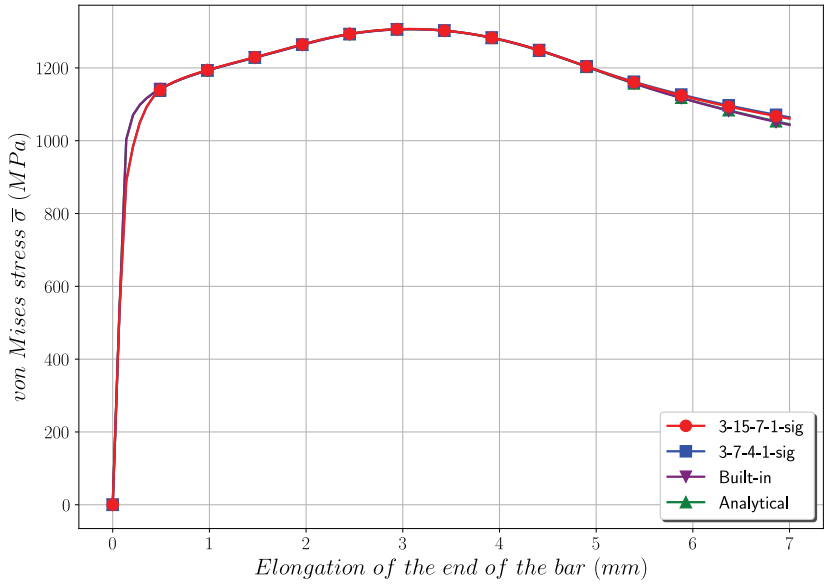

Fig. 9. Von Mises stress $\overline{\sigma}$ *vs.* displacement for the necking of a circular bar.

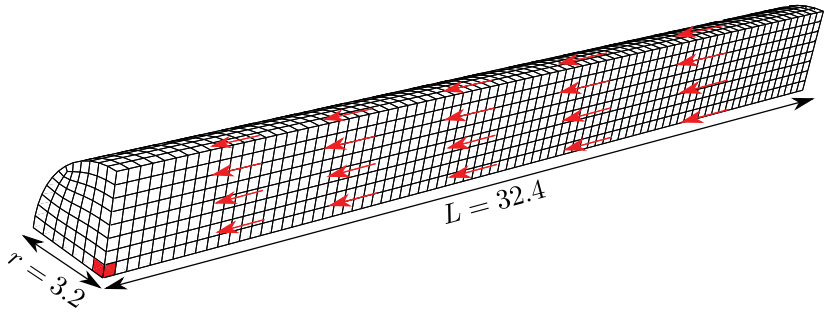

Fig. 10. Numerical model for the Taylor impact test.

As presented in Section 2 the neural network is based on two main parts defined in Sections 2.1 and 2.2 corresponding to the computation of the von Mises equivalent stress σ and the 3 derivatives $\partial\sigma/\partial\epsilon^p$, $\partial\sigma/\partial\dot{\epsilon}^p$ and $\partial\sigma/\partial T$ respectively. If we want to implement a 2 hidden layers ANN with a sig activation function containing m neurons in layer 1 and n neurons in layer 2, we must use Eq. (5) for the computation

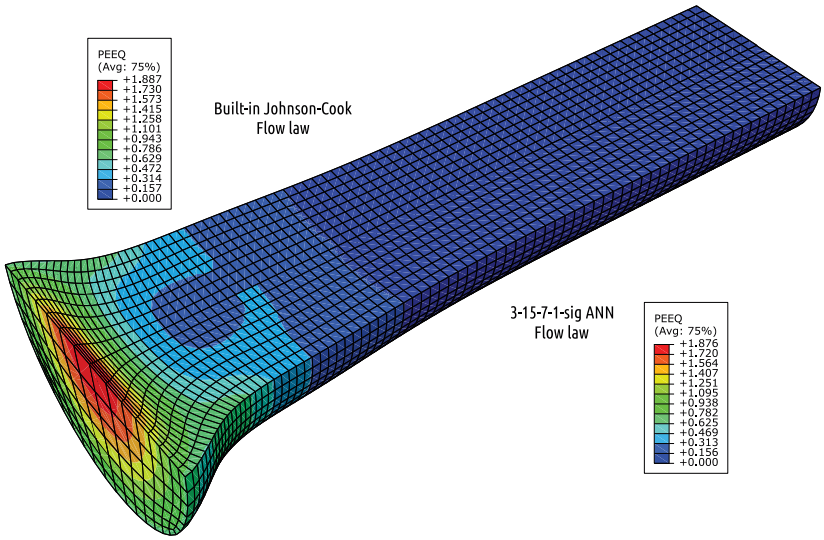

Fig. 11. Equivalent plastic strain $\overline{\varepsilon}^p$ contourplot for the Taylor impact test (upper side is the Built-in model and bottom side is the ANN 3-15-7-1-sig model).

of the stress and Eq. (17) for the derivatives. In order to optimize the numerical implementation a bit, and, as the computation of the flow stress and its derivatives share some common terms that will be stored during the computation for later use, the computation of Eqs. (5) and (17) is split into several sub-terms $\overline{z}^a$ to $\overline{z}^f$. So, starting from , defining the values of $\vec{x}$, one can write:

$$\begin{aligned} z_i^a &= \exp\left(-\sum_j \left(w_{ij}{}^{(1)} x_j\right) - b_i{}^{(1)}\right) && i \in [1,m],\ j \in [1,3] \\ z_i^b &= 1 + z_i^a && i \in [1,m] \\ z_i^c &= \exp\left(-\sum_j \left(w_{ij}{}^{(2)} / z_j^b\right) - b_i{}^{(2)}\right) && i \in [1,n],\ j \in [1,m] \end{aligned} \tag{23}$$

where z_i^a, z_i^c and z_i^c are three terms present in Eq. (17) corresponding to $\exp\left(-\overline{y}^{(1)}\right)$, $1+\exp\left(-\overline{y}^{(1)}\right)$ and $\exp\left(-\overline{y}^{(2)}\right)$ respectively. Then for the computation of the derivatives, we combine the values z_i^a, z_i^b and z_i^c to compute the whole equation (17) using:

$$\begin{aligned} z_i^d &= w_i z_i^c / (1+z_i^c)^2 && i \in [1,n] \\ z_i^e &= z_i^a / z_i^b && i \in [1,m] \\ z_i^f &= \sum_j \left(w_{ji}^{(2)} z_j^d\right) z_i^e && i \in [1,m],\ j \in [1,n] \end{aligned} \tag{24}$$

From those definitions, one can then write the output of the neural network using the following expression:

$$s = \sum_i \left(w_i / (1+z_i^c)\right) + b \qquad i \in [1,n] \tag{25}$$

And, the three derivatives s_i' are obtained from:

$$s_i' = \sum_j \left(w_{ji}^{(1)} z_j^f\right) \qquad i \in [1,3],\ j \in [1,m] \tag{26}$$

Finally, Eqs. (11) and (18) are used to compute the von Mises equivalent stress σ and the 3 derivatives $\partial\sigma/\partial\varepsilon^p$, $\partial\sigma/\partial\dot{\varepsilon}^p$ and $\partial\sigma/\partial T$ of the neural network from s and $\overline{s}'$ computed from Eqs. (25) and (26). As it is a straightforward implementation from Eq. (23) to (26) the Python interface uses loops to explicitly write all the matrices products in a FORTRAN subroutine as illustrated in Fig. 5. This later only reports a small part of the whole FORTRAN code to illustrate how it is implemented. The reader interested in the details of this implementation can refer to the Software Heritage archive website [39] giving access to the source code of this work.

The VUHARD subroutine is compiled using the GNU gfortran 9.3.0 and linked to the main Abaqus Explicit executable. All benchmarks tests have been solved using Abaqus Explicit 2021 on a Dell XPS 13 laptop running Ubuntu 20.04 64 bits with 16 GiB of Ram and one 4 core i7-10510U Intel Processor. All computations have been done using the double precision option of Abaqus, with parallel threads execution on two cores. In order to reduce the number of models presented hereafter, only the two last models presented in Table 2 are selected for the subsequent benchmark simulations.

4.2. Necking of a circular bar benchmark test

The necking of a circular bar test, already presented by Ponthot et al. [40], is useful to evaluate the performance of non-linear constitutive laws. An axisymmetric quarter model of the specimen, already presented in Ming et al. [4], is used, where dimensions of the specimen are reported in Fig. 6. We imposed a total displacement of 7 mm along the $\overline{z}$ axis on the left side of the specimen while the radial displacement of the same edge is supposed to remain zero. On the opposite side, the axial displacement is restrained while the radial displacement is free. The mesh consists of 400 CAX4RT elements (4-node bilinear displacement and temperature, reduced integration with hourglass control element) with a refined zone of 200 elements on the right side on 1/3 of the total length. The FEM model is a coupled temperature–displacement explicit model (this is a coupled thermal-stress analysis where the heat transfer and mechanical solutions are obtained simultaneously by an explicit coupling), and the total simulation time is set to $t = 0.01$ s.

Fig. 7 shows the von Mises stress contourplot $\overline{\sigma}$ of the deformed bar for two different models: the Built-In model (top side) and the ANN 3-15-7-1-sig model (bottom side). There is a very little difference between both models in terms of spatial distribution of stress and maximum value. The maximum stress is located into the center of the bar, so we have chosen to plot in Figs. 8 and 9 the evolution of the equivalent plastic strain $\overline{\varepsilon}^p$ and the von Mises stress $\overline{\sigma}$ for the central element of the specimen (the red element in the bottom right corner in Fig. 6). As reported in Figs. 8 and 9, the Built-In model, the Analytical model and both versions of the ANN model give almost the same results, except when the elongation is greater than 6 mm where the von Mises stress differs between the ANN models and the Built-in and Analytical ones as we will see further.

This is also confirmed in Table 3 reporting a comparison of the four models for two values of the displacement (3.5 mm and 7 mm, named *mid* and *end* respectively). From this later, we can see that the equivalent plastic strain $\overline{\varepsilon}^p$, the von Mises stress $\overline{\sigma}$ and the temperature T obtained with both ANN models are very close to the ones obtained by the Built-in and the Analytical models for *mid* displacement, while they differ a little for *end* displacement. This allows us to validate the proposed approach.

Table 3
Comparison of results for the necking of a circular bar benchmark for a displacement of 3.5 mm (mid) and 7 mm (end).

Model	Incr.	Time (s)	$\overline{\varepsilon}^p_{mid}$	$\overline{\sigma}_{mid}$ (MPa)	T_{mid} (°C)	$\overline{\varepsilon}^p_{end}$	$\overline{\sigma}_{end}$ (MPa)	T_{end} (°C)
3-7-4-1-sig	191 768	29.98	0.51	1293.81	182.01	2.16	1064.43	587.78
3-15-7-1-sig	194 432	38.54	0.51	1293.97	181.52	2.03	1060.50	587.29
Analytical	200 145	35.50	0.51	1293.59	182.47	2.16	1045.75	585.85
Built-In	199 474	28.71	0.51	1293.76	180.36	2.14	1043.19	587.66

One interesting result concerns the values of the equivalent plastic strain $\overline{\varepsilon}^p$ and the temperatures T reported in Table 3 for *end* displacement. It can be seen that, for *end* displacement, the values of the plastic strain reported in Table 3 are around 2.1 while the range of plastic strain used for training the model is $[0, 1]$. Furthermore, the maximum value of the temperature is around 587°C while the training range is $[20, 500]$. The proposed model is therefore able to extrapolate with a good accuracy some data out of the training range. It is obvious from Table 3 and Figs. 8 and 9 that all models give very closed results when parameters are within the training range, while the ANN models differ a little when parameters are far out of the training range (here, with an elongation greater than 6 mm so that $\overline{\varepsilon}^p > 1.7$). This shows that our ANN model is able, to a certain extent, and with the usual precautions insofar as neural networks are generally relatively faithful for interpolation but less so for extrapolation, to generalize a behavior out of the training range. If we want to reduce the gap between predicted and real values out of the training range, we have to enlarge the training ranges of the input variables if data is available.

Table 3 reports also the computing times and total number of increments computed after running the same model 10 times. One can see that computation time of the 3-7-4-1-sig model is equivalent to the one of the Built-in model and lower to the one of the Analytical model while there is an increase for the most complex ANN model. In the proposed approach, the ANN is used to compute the flow stress of the material within a Radial Return algorithm. It replaces the evaluation of the flow stress and its derivatives based on some analytical expressions. VUHARD implementation breaks the Built-in natural optimized algorithm used to compute the stresses by some FORTRAN subroutine call, data transfer, ... leading to increase of CPU time (see comparison between Built-in and Analytical CPU times). This is why we cannot have a reduction of computational time with regard to the Built-in model. We therefore have to compare CPU performance of ANN models with the Analytical one. Since the Johnson–Cook analytical behavior law is quite simple, the most complex ANN model is slower than the analytical one. But, in case of more complex behavior law, such as some Modified Johnson–Cook laws proposed by Zhou et al. [41], this tendency will be reversed since the analytical evaluation of the derivatives needed for the radial return algorithm to work becomes quite CPU intensive. We can also conclude from the results presented in Table 3 that the 3-7-4-1-sig model is sufficient to obtain valuable results with comparable CPU times with regard to the Built-in constitutive law.

4.3. Taylor impact benchmark test

The performance of the proposed ANN subroutine will now be validated under high deformation rate with the simulation of the Taylor impact test [42] where a cylindrical specimen is launched to impact a rigid target with a prescribed initial velocity $V_c = 287$ m/s. The height of the cylinder is 32.4 mm and the radius is 3.2 mm as reported in Fig. 10. The axial displacement is restrained on the left side of the specimen while the radial displacement is free (to figure a perfect contact without friction of the projectile onto the target). A 3D quarter model of the Taylor cylindrical specimen is meshed with 4 455 C3D8T elements (8-node trilinear displacement and temperature). The total simulation time for the Taylor impact test is $t = 80\,\mu$s.

Table 4
Comparison of results for the 3D Taylor impact test.

Model	Incr.	Time (s)	L_f (mm)	D_f (mm)	T (°C)	$\overline{\varepsilon}^p$
3-7-4-1-sig	6 344	63.08	26.52	11.18	584.28	1.83
3-15-7-1-sig	6 239	90.32	26.52	11.17	584.47	1.83
Analytical	6 419	71.71	26.53	11.19	585.64	1.84
Built-In	6 570	52.82	26.54	11.21	588.66	1.84

Fig. 11 shows the equivalent plastic strain contourplot of the deformed rod for two models: the Built-In model (upper side of the specimen) and the ANN-3-15-7-1-sig model (bottom side of the specimen). The distributions of the equivalent plastic strain are almost the same for both models. The maximum equivalent plastic strain $\overline{\varepsilon}^p$ is located into the center element of the model (the red element in Fig. 10) and the models give quite the same value as reported in Table 4 for $\overline{\varepsilon}^p$, T and the final dimensions of the specimen L_f (final length) and D_f (final diameter of the impacting face). Concerning the simulation times reported in Table 4, the same trends as those presented in Section 4.2 are noticeable in this test case. Again, the comparison of the numerical results validates the proposed approach and shows a very good level of correlation of the results.

5. Summary and conclusions

In this paper, an Artificial Neural Network based framework has been proposed to model the non-linear flow law $\sigma^y(\varepsilon^p, \dot{\varepsilon}^p, T)$ with its application to a 42CrMo4 steel and a constitutive behavior of type Johnson–Cook. The general architecture of the multilayer perceptron neural network was presented with a focus on the evaluation of the derivatives of the flow stress with respect to the plastic strain, the plastic strain rate and the temperature necessary to implement a VUHARD user routine in the finite element code Abaqus Explicit, without these quantities having been learned by the network according to the classical supervised training scheme. The evaluation of these derivatives for 1 or 2 hidden layers and 2 types of activation function has been presented in detail as well as the comparison of the accuracy of this evaluation with a reference solution based on the use of the Johnson–Cook flow law. The results obtained showed an excellent ability to evaluate the flow stress and a very good ability to evaluate the derivatives by the neural network. After numerical implementation of the neural network in the Abaqus code, the test cases used showed the good behavior of the proposed approach in the context of the numerical simulation of the necking of a circular bar and a Taylor impact test.

CRediT authorship contribution statement

Olivier Pantalé: Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Pierre Tize Mha:** Acquisition of data, Writing – review & editing. **Amèvi Tongne:** Analysis and/or interpretation of data, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

All authors approved the final version of the manuscript.

References

[1] C.Y. Gao, FE realization of a thermo-visco-plastic constitutive model using VUMAT in ABAQUS/Explicit program, in: Computational Mechanics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, p. 301.

[2] G. Jansen van Rensburg, S. Kok, Tutorial on state variable based plasticity: An Abaqus UHARD subroutine, in: Eighth South African Conference on Computational and Applied Mechanics - SACAM2012, Johannesburg, 2012.

[3] N. Duc-Toan, B. Tien-Long, J. Dong-Won, Y. Seung-Han, K. Young-Suk, A modified Johnson–Cook model to predict stress-strain curves of boron steel sheets at elevated and cooling temperatures, High Temp. Mater. Process. 31 (2012) 37–45.

[4] L. Ming, O. Pantalé, An efficient and robust VUMAT implementation of elasto-plastic constitutive laws in Abaqus/Explicit finite element code, Mech. Ind. 19 (3) (2018) 308.

[5] D. Versino, A. Tonda, C.A. Bronkhorst, Data driven modeling of plastic deformation, Comput. Methods Appl. Mech. Engrg. 318 (2017) 981–1004.

[6] G. Bomarito, T. Townsend, K. Stewart, K. Esham, J. Emery, J. Hochhalter, Development of interpretable, data-driven plasticity models with symbolic regression, Comput. Struct. 252 (2021).

[7] H. Park, M. Cho, Multiscale constitutive model using data– driven yield function, Composites B 216 (2021).

[8] A. Nassr, A. Javadi, A. Faramarzi, Developing constitutive models from EPR-based self-learning finite element analysis, Int. J. Numer. Anal. Methods Geomech. 42 (3) (2018) 401–417.

[9] C. Mattmann, Machine Learning with Tensorflow, O'REILLY MEDIA, S.l., 2020.

[10] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366.

[11] M.L. Minsky, S. Papert, Perceptrons; an Introduction To Computational Geometry, MIT Press, 1969.

[12] M.B. Gorji, M. Mozaffar, J.N. Heidenreich, J. Cao, D. Mohr, On the potential of recurrent neural networks for modeling path dependent plasticity, J. Mech. Phys. Solids 143 (2020).

[13] M. Jamli, N. Farid, The sustainability of neural network applications within finite element analysis in sheet metal forming: A review, Measurement 138 (2019) 446–460.

[14] P. Jiao, A.H. Alavi, Artificial intelligence-enabled smart mechanical metamaterials: Advent and future trends, Int. Mater. Rev. (2020) 1–29.

[15] J. Ghaboussi, J.H. Garrett, X. Wu, Knowledge-based modeling of material behavior with neural networks, J. Eng. Mech. 117 (1) (1991) 132–153.

[16] J. Ghaboussi, D.A. Pecknold, M. Zhang, R.M. Haj-Ali, Autoprogressive training of neural network constitutive models, 1998, p. 22.

[17] J. Ghaboussi, D. Sidarta, New nested adaptive neural networks (NANN) for constitutive modeling, Comput. Geotech. 22 (1) (1998) 29–52.

[18] Y. Lin, J. Zhang, J. Zhong, Application of neural networks to predict the elevated temperature flow behavior of a low alloy steel, Comput. Mater. Sci. 43 (4) (2008) 752–758.

[19] A. Javadi, M. Rezania, Intelligent finite element method: An evolutionary approach to constitutive modeling, Adv. Eng. Inform. 23 (2009) 442–541.

[20] Z. Lu, Q. Pan, X. Liu, Y. Qin, Y. He, S. Cao, Artificial neural network prediction to the hot compressive deformation behavior of Al–Cu–Mg–Ag heat-resistant aluminum alloy, Mech. Res. Commun. 38 (3) (2011) 192–197.

[21] H.R. Ashtiani, P. Shahsavari, A comparative study on the phenomenological and artificial neural network models to predict hot deformation behavior of AlCuMgPb alloy, J. Alloys Compd. 687 (2016) 263–273.

[22] U. Ali, W. Muhammad, A. Brahme, O. Skiba, K. Inal, Application of artificial neural networks in micromechanics for polycrystalline metals, Int. J. Plast. 120 (2019) 205–219.

[23] M. Stoffel, F. Bamer, B. Markert, Artificial neural networks and intelligent finite elements in non-linear structural mechanics, Thin-Walled Struct. 131 (2018) 102–106.

[24] M. Stoffel, F. Bamer, B. Markert, Neural network based constitutive modeling of nonlinear viscoplastic structural response, Mech. Res. Commun. 95 (2019) 85–88.

[25] X. Li, C.C. Roth, D. Mohr, Machine-learning based temperature- and rate-dependent plasticity model: Application to analysis of fracture experiments on DP steel, Int. J. Plast. 118 (2019) 320–344.

[26] X. Huang, Y. Zang, B. Guan, Constitutive models and microstructure evolution of Ti-6Al-4V alloy during the hot compressive process, Mater. Res. Express 8 (1) (2021).

[27] D.W. Abueidda, S. Koric, N.A. Sobh, H. Sehitoglu, Deep learning for plasticity and thermo-viscoplasticity, Int. J. Plast. 136 (2021).

[28] F. Masi, I. Stefanou, P. Vannucci, V. Maffi-Berthier, Thermodynamics-based Artificial Neural Networks for constitutive modeling, J. Mech. Phys. Solids 147 (2021).

[29] J.C. Knight, T. Nowotny, Larger GPU-accelerated brain simulations with procedural connectivity, Nat. Comput. Sci. 1 (2) (2021) 136–142.

[30] D. Specht, A general regression neural network, IEEE Trans. Neural Netw. 2 (6) (1991) 568–576.

[31] T. Nguyen-Thien, T. Tran-Cong, Approximation of functions and their derivatives: A neural network implementation with applications, Appl. Math. Model. 23 (9) (1999) 687–704.

[32] G.R. Johnson, W.H. Cook, A constitutive model and data for metals subjected to large strains, high strain rates, and high temperatures, in: Proceedings 7th International Symposium on Ballistics, The Hague, 1983, pp. 541–547.

[33] O. Dalverny, S. Caperaa, O. Pantalé, C. Sattouf, Identification de lois constitutives et de lois de frottement adaptées aux grandes vitesses de sollicitation, J. de Physique IV - Proc. 12 (11) (2002) 275–282.

[34] C. Sattouf, O. Pantalé, S. Caperaa, A methodology for the identification of constitutive and contact laws of metallic materials under High Strain Rates, in: Advances in Mechanical Behaviour, Plasticity and Damage, Elsevier, Tours, France, 2000, pp. 621–626.

[35] C. Sattouf, Caractérisation en dynamique rapide du comportement de matériaux utilisés en aéronautique, (Ph.D. thesis), Toulouse, INPT, 2003.

[36] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: A system for large-scale machine learning, in: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, in: OSDI'16, USENIX Association, USA, 2016, pp. 265–283.

[37] D.P. Kingma, J. Lei, Adam: A Method for Stochastic Optimization, 2015, p. 15.

[38] J.C. Simo, T.J.R. Hughes, Computational Inelasticity, in: Interdisciplinary Applied Mathematics, Springer, New York, 1998.

[39] O. Pantalé, ANN VUHARD repository, 2021, Software Heritage archive. ANN-FEAD-2021.

[40] J.P. Ponthot, Unified stress update algorithms for the numerical simulation of large deformation elasto-plastic and elasto-viscoplastic processes, Int. J. Plast. (2002) 36.

[41] Q. Zhou, C. Ji, M.-y. Zhu, Research on several constitutive models to predict the flow behaviour of GCr15 continuous casting bloom with heavy reduction, Mater. Res. Express 6 (12) (2020).

[42] G.I. Taylor, The testing of materials at high strain rates of loading, J. Inst. Civ. Eng. 26 (8) (1946) 486–519.