Miguel Gonçalves Gaspar

Desenvolvimento de um modelo de comportamento elastoplástico através de inteligência artificial

Development of a model of elastoplastic behavior through artificial intelligence

Miguel Gonçalves Gaspar

Desenvolvimento de um modelo de comportamento elastoplástico através de inteligência artificial

Development of a model of elastoplastic behavior through artificial intelligence

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob a orientação científica de António Gil D'Orey de Andrade Campos, Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Rui António da Silva Moreira
Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Hugo Filipe Pinheiro Rodrigues
Professor Adjunto do Departamento de Engenharia Civil do Instituto Politécnico de Leiria, Escola Superior de Tecnologia e Gestão (arguente principal)

Prof. Doutor António Gil D'Orey Andrade Campos
Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

É este o momento de tornar público o meu profundo agradecimento a todos os que, nos últimos cinco anos, me acompanharam no meu percurso universitário

Ao orientador desta dissertação o Professor Gil Campos, pela orientação prestada, pelo seu incentivo, disponibilidade e apoio que sempre demonstrou. Aqui lhe exprimo a minha gratidão.

Um grande obrigada à minha familia, em especial à minha mãe, ao meu pai e à minha irmã, pelo eterno apoio e por todas as oportunidades que me proporcionaram.

Gostaria também de agradecer à Joana Mota pela imensa paciência que sempre demonstou ter em me aturar.

Agradeço ao António, ao Gonçalo, ao Francisco, ao Filipe e a todos os meus amigos que sempre me acompanharam desde infância, mesmo estudando e vivendo em universidades e cidades diferentes.

Por fim, quero Agradecer ao Carneiro, à Sara, à Carmen, ao Barros, ao Miguel e ao Nunes, pelo gratificante percurso de cinco anos e pelas longas e extensas discussões em inúmeros trabalhos de grupo.

Palavras-chave

Inteligência Artificial, Aprendizagem Computacional, Aprendizagem Supervisionada, Regressão, Redes Neuronais, Modelos Constitutivos, Algoritmos de Otimização, FEA, Modelação Preditiva, *User Subroutines*, Comportamento Elastoplástico

Resumo

Nos últimos anos, tem havido enormes avanços na precisão e capacidades preditivas de ferramentas para a simulação de materiais. A modelação preditiva tornou-se numa ferramenta poderosa que também pode agregar um grande valor por meio de aplicações e inovações para a indústria global. A simulação das operações de conformação, particularmente usando o método dos elementos finitos, é claramente dependente da precisão dos modelos constitutivos. Nos últimos anos, várias metodologias foram desenvolvidas para melhorar a precisão de modelos constitutivos através de metodologias de identificação e calibração de parâmetros. No entanto, independentemente da eficácia dos métodos de calibração, a precisão de um modelo constitutivo é sempre restrita à sua formulação matemática predefinida. Adicionalmente, usando formulações elastoplásticas conhecidas, é impossível reproduzir os fenómenos do comportamento de materiais se estes comportamentos não forem eficazmente formulados matematicamente.

Recentemente, as técnicas de inteligência artificial (IA) tornaram-se mais robustas e complexas. Este campo estabeleceu o objetivo ambicioso de tornar as máquinas aparentemente ou genuinamente inteligentes. O sub-campo da inteligência artificial conhecido como aprendizagem computacional tenta fazer com que os computadores aprendam com as observações. Os algoritmos de aprendizagem computacional são ferramentas gerais que podem ser adaptadas a um grande número de problemas, incluindo a previsão da relação tensão-deformação do material.

Este trabalho propõe modelar o comportamento de um material metálico utilizando técnicas de aprendizagem computacional (ML) e utilizar este ML na modelação de simulações. Inicialmente, o modelo ML é projetado e treinado usando um modelo de elastoviscoplasticidade em estado plano de tensão de forma a avaliar a sua eficácia na substituição de modelos clássicos. Diferentes topologias ML e técnicas de otimização são usadas para treinar o modelo. Em seguida, o modelo IA é introduzido num código de análise de elementos finitos (FEA), como *user subroutine*, e a sua concretização em simulações de conformação é avaliada. A substituição de formulações clássicas por técnicas de IA para a definição do comportamento do material é analisada e discutida.

Keywords Artificial Intelligence, Machine Learning, Supervised Learning, Regression, Neural-Networks, Constitutive Models, Optimization Algorithms, FEA, Predicitive Modeling, User Subroutines, Elastoplastic Behaviour

Abstract

In the past few years, there has been tremendous advances in the accuracy and predictive capabilities of tools for the simulation of materials. Predictive modeling has now become a powerful tool that can also deliver real value through application and innovation to the global industry. Simulation of forming operations, particularly using the finite element method, is clearly dependent on the accuracy of the constitutive models. In the last years, several methodologies were developed to improve the accuracy of constitutive models through parameter identification and calibration methodologies. However, independently of the efficacy of the calibration methods, the accuracy of a constitutive model is always constrained to its predefined mathematical formulation. Additionally, using known elastoplastic formulations, it is impossible to reproduce the material phenomena if these phenomena are not formulated mathematically.

In the past several years, artificial intelligence (AI) techniques have become more robust and complex. This field has set the ambitious goal of making machines either seemingly or genuinely intelligent. The sub-field of artificial intelligence known as machine learning attempts to make computers learn from observations. Machine-learning algorithms are general tools that can be fitted to a vast number of problems, including predicting the stress-strain relationship of the material.

This work proposes to model the behavior of a metal material using machine-learning (ML) techniques and use this ML in forming simulations. Initially, the ML model is designed and trained using a known plane stress elastoviscoplasticity model to evaluate its competence to replace classical models. Different ML topologies and optimization techniques are used to train the model. Then, the AI model is introduced into a finite element analysis (FEA) code, as a user subroutine, and its attainment in forming simulations is evaluated. The replacement of classical formulations by AI techniques for the material behavior definition is analysed and discussed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this past few years, there has been tremendous advances in the accuracy and predictive capabilities of tools for the simulation of materials. Predictive modeling has now become a powerful tool which can also deliver real value through application and innovation to the global industry. However, the fast pace which new complex materials are developed to assist the objective of industrial designers leads to the increasing necessity for non-linear analysis of materials which are characterized by material models or constitutive equations.

Finite Element Method (FEM) is a numerical method that has been widely used as a powerful tool in the analysis of engineering and mathematical physics problems, such as structural analysis. In FEM, the behavior of the material is defined with some constitutive relationships. Therefore, the selection of an appropriate constitutive model, capable of adequately describing the behavior of the material, plays a significant role in the reliability and precision of the numerical predictions. For a non-linear behavior, the hardest problem that the technique commonly face is to reduce the error between the numerical prediction by the model and the experimental values. This can only be improved by increasing the number of parameters, many of which have no physical meaning. However, the decrease of model error does not tackle the substance of the problem since all models are limited by the capability of their mathematical description, since the model is written explicitly.

In the past years, artificial intelligence techniques have become more robust and complex. This field has set the ambitious goal of making machines either seemingly or genuinely intelligent. The sub-field of artificial intelligence known as machine learning attempts to make computers learn from observations. While statistical time series models are specialized for their task, machine learning algorithms are general tools that can be fitted to a vast number of problems, including predicting the relationship between the stress and strain in the material.

A standard neural network (NN) mimics how brain neurons works. The network is made of enumerous simple connected processors, called neurons, each producing a sequence of real-valued activations. These neurons can be grouped as input, hidden and output layers, as it can be seen in figure 1.1. Typically, a neural network is initially trained and fed with large amounts of data. Training consists on providing input and highlighting to the network what the output should be. Then, the hidden neurons adapt their values to better fit the training data and, as result, give predictions with untrained inputs. The training phase uses optimization techniques to find efficiently the neuron's values.

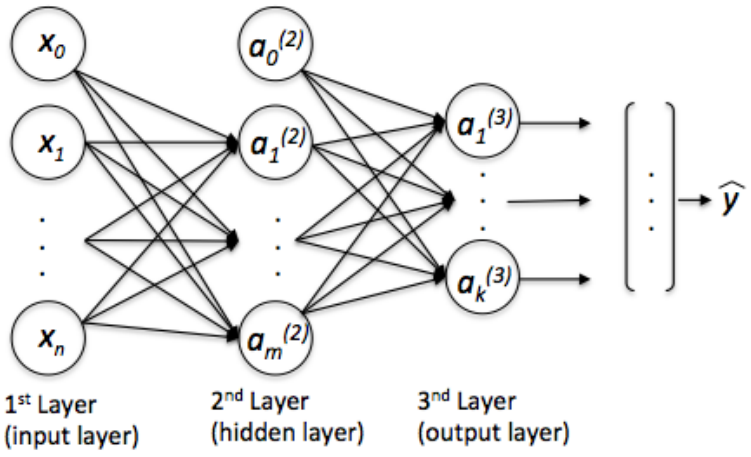


Figure 1.1: Simple representation of an artificial neural network.

Considering the machine learning advantages and capabilities, a neural network model could be incorporated in the finite element analysis (FEA) code as a replacement to the constitutive material model.

1.1 Related Work

In order to provide data to the training process of machine learning algorithms, some authors used experimental data yet others used computed data. In [1, 2, 3], the authors used experimental data. On [1], data from soils is used. Other approach is given in [2], where the authors analyzed results from a bundle of super-conduction cables used in a nuclear power plant as an experimental investigation of its mechanical properties. Another interesting factor to have in mind is concerning noise in the experimental measurements, which was presented in [3]. They conclude that the constitutive relations could be modeled regardless of the noise level when sufficient data are available. However, on a first step, the data used to train the machine learning model is generated using an analytical formulation, for instance, the Chaboche viscoplasticity model, in resemblance with the work made on [4].

The next step is to chose an appropriate model to train the dataset. In [4], a multilayer feedfoward neural network is presented, as a universal function approximator for any bounded square integrable function of many variables. However, the authors in [4] chose the same number of training and test sets which can substantially decrease the reliability of their implemented model. Other work, such as [5] implemented a Nested Adaptive Neural Network (NANN), which is a variation of standard multi-layer neural network. The network consists of a total of four layers, one input layer (strain vector and state variables), two hidden layers and an output layer (stress vector). An initial number of nodes to each hidden layer are assigned, then the number of nodes can vary to better fit the training data. On [2, 3], both authors use a back-propagation neural network, but in [3] the algorithm is modified and it is only based on the error of the strain energy and the external work without the need of stress data.

The implementation of these trained models whether to test or to use on finite element analysis was poorly or not described in [4, 3]. Nevertheless, in [5], the authors report the issues related to this numerical implementation. The trained model is implemented in the widely used general-purpose finite element code Abaqus , using the user defined material module UMAT capability. Also in [1], a similar approach was made but the authors does not specify the framework used for the so called NeuroFEM. Besides the software part, in [2], the constitutive law represented by ANN is described analytically inside a FE code. The authors rewrite its standard equations and explain how this application is possible.

The implementation of the material behavior description is schematically represented in 1.2.

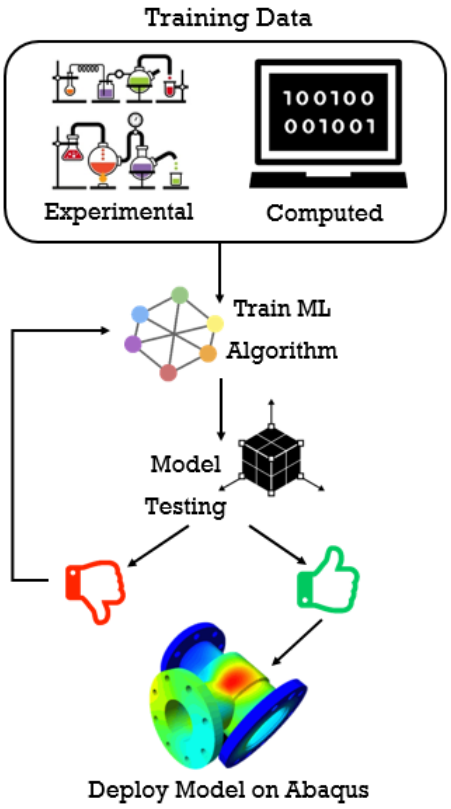


Figure 1.2: Machine Learning approach in material characterization.

Recently the use of artificial neural networks to reproduce constitutive models are presented in different projects [6, 7, 8, 9, 10, 11]. In all of these, the modelling of hot deformation behavior of a specific material using artificial neural networks are described. However, in [11] the author develops a constitutive model from an evolutionary polynomial regression-based self-learning finite element analysis. Some authors [12, 13, 14] use artificial neural networks to predict the failure of glass/epoxy composite pipes, fatigue life assessment of in-service road bridge decks and stress hotspots. In [15], the author propose the predicting effect of cooling rate on the mechanical properties of glass fiber-polypropylene composites.

1.2 Objectives

The objective is to develop a model for elastoplastic behavior using artificial intelligence techniques. For this purpose, it is necessary to understand such techniques, as well as the material constitutive models in order to generate virtually experimental tests. After the training process using the virtually experimental data, the proposed model to be developed needs to be validated in non-trained strain fields.

Once the machine learning models are validated, an implementation and analysis of the proposed model on a FEA code is intended, as well as the validation of such implementation. Finally, an application of the system on a more practical engineer problem is analyzed and discussed.

1.3 Reading Guide

This dissertation is composed of 6 chapters, including the present one, in order to meet the objectives described above in a more organized way. Those chapters are arranged as follows:

1. **Introduction -** The current chapter in which is presented a brief introduction, a description of the problem *per se*, related projects and an enumeration of the objectives.

2. **Artificial Neural Networks -** Introduces and describes the mathematical formulation behind neural networks, which software is used and some machine learning methods for model evaluation and selection. It is also presented a way to persist the trained model without having to retrain every time it needs to perform predictions.

3. **Training -** The aim in this chapter is to develop trained neural networks to reproduce the material behavior for the chosen constitutive model in 1D and 2D. Thus, the constitutive models are described and virtually experimental tests are generated. Then, it focus on the training process and subsequent validation for both models with non-trained strain range.

4. **Implementation and analysis of the ANN-model on a FEA code -** Introduces the steps required to perform a simulation in FEA using user-subroutines. Then, the communication between the user-subroutine and the machine learning model is described, as well as the tests and results to validate this implementation.

5. **Testing machine learning model on a complex geometry -** Knowing that the implementation is validated, it was tested the trained neural network on the tensile test of the adapted butterfly specimen.

6. **Conclusions and Future Work -** Presents conclusions of the developed machine learning models and some suggestions for future work in this field of study.

Chapter 2

Artificial Neural Networks

2.1 Mathematical Formulation

Considering a simple neural network (figure 2.1) with an input layer of three units, a hidden layer with four units and an output layer with one unit, the calculations necessary to get the predictions ($\hat{\boldsymbol{y}}$) are described below:

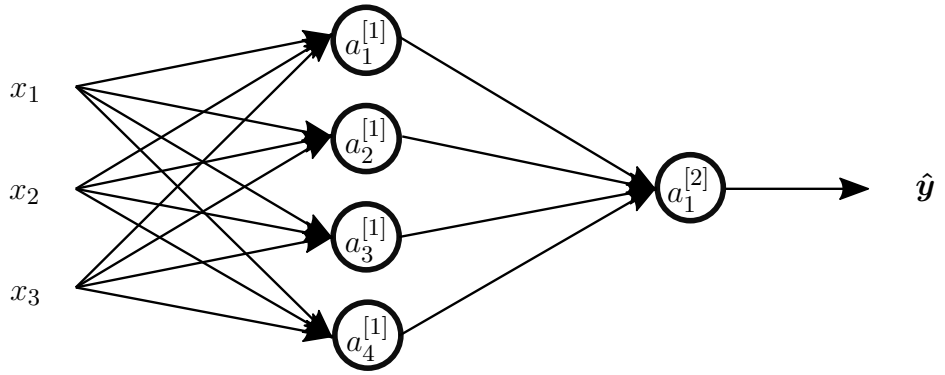
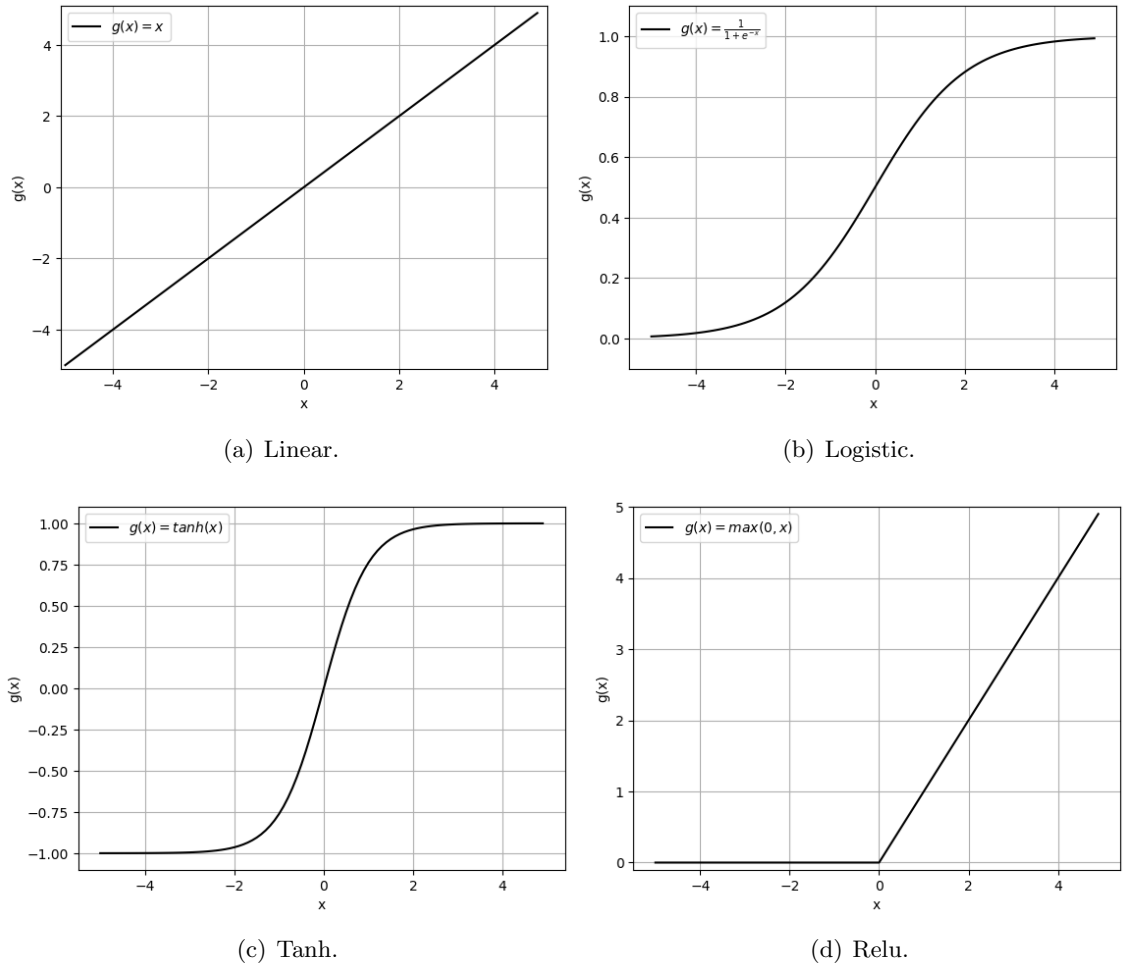


Figure 2.1: Artificial neural network with one hidden layer.

$$a_1^{[1]} = g(\Theta_1^{[1]T}\boldsymbol{x} + b_1^{[1]}), \tag{2.1}$$
$$a_2^{[1]} = g(\Theta_2^{[1]T}\boldsymbol{x} + b_2^{[1]}), \tag{2.2}$$
$$a_3^{[1]} = g(\Theta_3^{[1]T}\boldsymbol{x} + b_3^{[1]}), \tag{2.3}$$
$$a_4^{[1]} = g(\Theta_4^{[1]T}\boldsymbol{x} + b_4^{[1]}), \tag{2.4}$$
$$\hat{\boldsymbol{y}} = a_1^{[2]} = g(\Theta_1^{[2]T}\boldsymbol{a}^{[1]} + b_1^{[2]}), \tag{2.5}$$

where g() is the activation function, a_i^j is the activation of unit i in layer j, Θ_i^j is the matrix of weights controlling function mapping from layer j to layer j+1 of unit i and b_i^j is the bias.

The activation functions introduce linear or non-linear properties to the neural network. Their main purpose is to convert an input signal of a node to an output signal. Then, the output signal is used as an input in the next layer. Figure 2.2 illustrates some activation functions discussed in [16].



(a) Linear.

(b) Logistic.

(c) Tanh.

(d) Relu.

Figure 2.2: Activation functions used for artificial neural networks.

In this example, the activation nodes are calculated using a 4*3 matrix of parameters. Each row of the parameters are aplied to the inputs to obtain the value for one activation node. The hypothesis output is applied to the sum of the values of the activation nodes, which have been multiplied by another parameter matrix $\Theta^{[2]}$ containing the weights for the second layer of nodes.

A simpler forward neural network representation from equations 2.1:

$$\boldsymbol{a}^{[1]} = g(\Theta^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}), \tag{2.6}$$

$$\boldsymbol{a}^{[2]} = g(\Theta^{[2]}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}), \tag{2.7}$$

$$\hat{\boldsymbol{y}} = \boldsymbol{a}^{[2]}. \tag{2.8}$$

Training a neural network consists on finding the best parameters (Θ and $\boldsymbol{b}$) in order

to better represent the relationship between inputs and outputs. This result in a search for minimizing the cost function. For neural networks, the cost function is defined as:

$$J(\Theta, \boldsymbol{b}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[\boldsymbol{y}_k^{(i)}\log(\hat{\boldsymbol{y}}_k) + (1-\boldsymbol{y}_k^{(i)})\log(1-\hat{\boldsymbol{y}}_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{j,i}^{(l)})^2, \tag{2.9}$$

where L is the total number of layers in the network, s_l is the number of units, K is the number of output units, $\hat{\boldsymbol{y}}_k$ is the hypothesis for the k^{th} output and λ is the regularization factor.

In the training process, the regularization factor (λ) is responsible for penalize the parameters (Θ and $\boldsymbol{b}$) update.

With regard to the update of Θ and $\boldsymbol{b}$, gradient descent can be used as optimization algorithm. The partial derivatives are represented as:

$$d\Theta^{[1]} = \frac{dJ}{d\Theta^{[1]}}, \tag{2.10}$$

$$d\boldsymbol{b}^{[1]} = \frac{dJ}{d\boldsymbol{b}^{[1]}}, \tag{2.11}$$

$$d\Theta^{[2]} = \frac{dJ}{d\Theta^{[2]}}, \tag{2.12}$$

$$d\boldsymbol{b}^{[2]} = \frac{dJ}{d\boldsymbol{b}^{[2]}}. \tag{2.13}$$

Then, the parameters update is perform:

$$\Theta^{[1]} := \Theta^{[1]} - \alpha d\Theta^{[1]}, \tag{2.14}$$

$$\boldsymbol{b}^{[1]} := \boldsymbol{b}^{[1]} - \alpha d\boldsymbol{b}^{[1]}, \tag{2.15}$$

$$\Theta^{[2]} := \Theta^{[2]} - \alpha d\Theta^{[2]}, \tag{2.16}$$

$$\boldsymbol{b}^{[2]} := \boldsymbol{b}^{[2]} - \alpha d\boldsymbol{b}^{[2]}. \tag{2.17}$$

$$\tag{2.18}$$

Where the parameter α is the learning rate.

Gradient descent [17] is a first-order iterative optimization algorithm for finding the minimum of a function. In this case, it is to converge the parameters. Although, there are more robust and complex optimization algorithms, such as:

- Stochastic gradient descent [17], a stochastic aproximation of gradient descent;
- Adam [17], an extension to stochastic gradient descent;
- Limited-memory BFGS [18, 19].

In addiction, the back propagation algorithm is generally used. It consists on the repeated application of the error calculation used for gradient descent, but it is repeatedly applied in the reverse order, starting from output layer towards input layer. A more detailed explanation can be found in [20].

2.2 Topologies

Neural networks can be arranged in various forms and topologies. The necessity of improving the existing topologies led to the existence of different neural networks and how they work. In [21], the author present a map to navigate between emerging architectures and approaches to neural networks, which some examples are displayed on figure 2.3.

Feed Forward (FF) neural networks are defined by having all of their nodes fully connected, the activation flows from input layer to output, without back loops. These networks have only one hidden layer between input and output. Deep feed forward neural networks have the same principle as FF networks but with more than one hidden layer and uses back propagation to calculate the gradient to adjust weights. Recurrent Neural networks have the same basic principle as FF but the connections between nodes form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for a period of time. These type of networks can use their internal state to process sequences of inputs, making them applicable for unsegmented task like connected handwriting recognition or speech recognition. The last example shown in figure 2.3 are the Long/Short Term Memory networks, an extension of recurrent neural networks. A more detailed explanation can be found in [22].

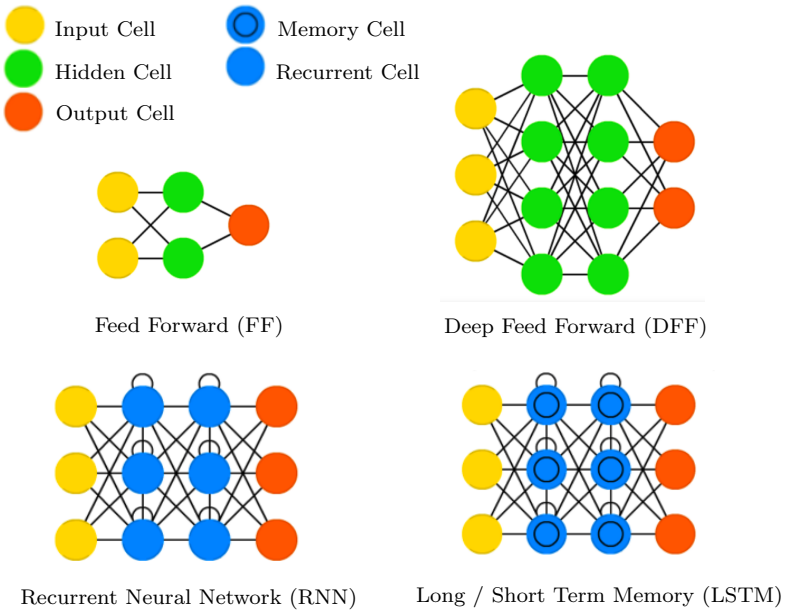


Figure 2.3: Examples of different neural network topologies.

2.3 Software

There are several open source libraries to implement machine and deep learning models. The list is large and Tensorflow [23], Pytorch [24], Keras [25] and Scikit-learn [26] are some examples. Selecting one technology from the options previous refereed can be a difficult task. First, due to the low size and complexity of the datasets, that is used for training, the best solution is a framework with an easy to use and debug interface. A major advantaged of this type of framework is having some internal methods for automatically search the best

hyper-parameters. With this in mind, Scikit-Learn is chosen.

2.3.1 Models

In Scikit-Learn ecosystem, the `neural_network` module includes three models based on neural networks. One for unsupervised learning, called Bernoulli Restricted Boltzmann Machine, and other two for supervised learning, the Multi-layer Perceptron Classifier and the Multi-layer Perceptron Regressor. Note that the scope of this work is not to explain in detail the mathematical formulations behind these models. The possibility of using an unsupervised model was put aside because each example on the dataset consists on a pair of an input object and a desired output value. Then, the Multi-layer perceptron regressor(MLPRegressor) module was selected because the goal in this work is to have an output on a set of continuous values and not a classification problem.

2.3.2 Pre-processing data

For regression estimators it's common use to standard scale the dataset. The result of standardization (or Z-score normalization) is that the features and the targets will be rescaled so that they have the properties of a standard normal distribution with zero mean and unit variance. This is done using $Scikit-Learn$'s class called `StandardScaler` from preprocessing module [27]. From this class the `fit` method is used which compute the mean and standard deviation to be used for later scaling. The `transform` method which perform standardization by centering and scaling. The `inverse_transform`, which transforms the dataset to it's initial state, is also used.

2.3.3 Model Evaluation and Selection

To quantify the performance of predictions for a given machine learning model, it is used metrics. For regression problems, Scikit-Learn provides mean square error [28], a risk metric corresponding to the expected value of the squared (quadratic) error or loss. Where $\hat{\boldsymbol{y}}_{\boldsymbol{i}}$ is the predicted value of the i-th sample, and $\boldsymbol{y_i}$ is the corresponding true value, then the mean squared error (MSE) estimated over $n_{samples}$ is computed by the following equation:

$$MSE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (\boldsymbol{y}_i - \hat{\boldsymbol{y}})^2. \tag{2.19}$$

It also provides the R^2 score, the coefficient of determination [29]. It provides a measure of how well future samples are likely to be predicted by the model. It's mathematical formulation is given by:

$$R^2(\boldsymbol{y}, \hat{\boldsymbol{y}}) = 1 - \frac{\sum_{i=0}^{n_{samples}-1} (\boldsymbol{y_i} - \hat{\boldsymbol{y}})^2}{\sum_{i=0}^{n_{samples}-1} (\boldsymbol{y_i} - \bar{\boldsymbol{y}})^2}, \tag{2.20}$$

where $\bar{\boldsymbol{y}}$ is described as:

$$\bar{\boldsymbol{y}} = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} \boldsymbol{y_i}. \tag{2.21}$$

Selecting a model or adapting it's parameters for machine learning tasks is often referred as tunning hyper-parameters.

Hyper-parameters can be divided into two groups: one containing the variables responsible for determine the network structure (e.g. number of hidden units for each hidden layer), and other with variables which establish how the network is trained (e.g. optimization algorithms, activation functions, learning rate, regularization factor). As shown in [30], the combinations of all these variables results in a search for the best parameters. Scikit-Learn have built-in functions (`grid_search`) which allow to perform this search. Documentation can be found in [31].

Scikit-Learn provides a function to test a range of hyper-parameters using cross-validation [32]. This method consists into splitting the dataset into k folds. For each k-fold the model is trained on k-1 folds of the dataset, then the model is tested to check the effectiveness for the correspondent fold. This process is repeated until each of the k-folds has served as test set. The overall error is computed as the average of each fold error. This method is repeated through the range of chosen hyper-parameters. An example of this method is illustrated in figure 2.4, where $k = 5$.

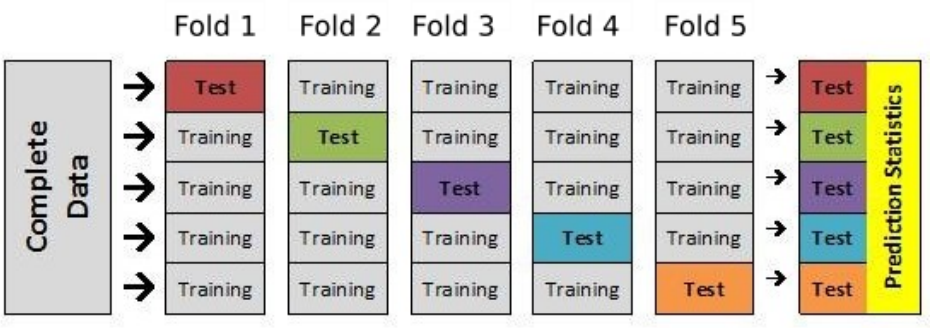


Figure 2.4: Cross-validation method using 5 folds [33].

Scikit-Learn also provides the `validation_curve` [34] function to test different combinations for a single hyper-parameter using cross-validation. It is calculated the validation and training score to visually analyze the model performance. The `learning curve` [34] is also used to show the validation and training scores of an estimator for varying numbers of training samples.

2.3.4 Model Persistence

In order to use the machine learning model, it is desirable to have a way to persist the model without having to retrain every time it needs to perform predictions. It is possible to save a model in Scikit-Learn by using `joblib` [35] class from `sklearn.externals` module. To save and load the model, `joblib.dump` and `joblib.load` were used, respectively. These functions were also used to save and load the two `StandardScaler` classes described in section 2.3.2. One transforms the features (scaler_x) and other to inverse transform the predictions (scaler_y). This flow of actions results in three pickle files: estimator.pkl, scaler_x.pkl and scaler_y.pkl.

Chapter 3

Training

3.1 Creating Data for Machine Learning training

3.1.1 Chaboche Constitutive Model in 1D

The goal of this section is to compute an explicit viscoplastic material model in 1D. For that purpose, the computation of structures beyond the yield limit requires hardening properties and material's viscosity. There are some mathematical constitutive equations to describe the elasto-viscoplastic material behavior, but only Chaboche's model is in the scope of this research.

First, the stress-strain behavior is often studied by separating it's elastic properties from overall behaviors (inelastic properties). So, the sum of elastic strain with inelastic strain returns the total strain under uniaxial loading condition:

$$\varepsilon = \varepsilon^{\mathrm{e}} + \varepsilon^{\mathrm{in}}. \tag{3.1}$$

A material in it's elastic behavior, the stress can be computed between the linear relationship with strain, using Young's Modulus as slope:

$$\sigma = E\varepsilon^{\mathrm{e}}. \tag{3.2}$$

In the unified theory capable of report cyclic loading and viscous behavior, the time-dependent effect is unified with the plastic deformations as a viscoplastic term, where ϵ^{e} and $\varepsilon^{\mathrm{vp}}$ represent the elastic and viscoplastic behavior:

$$\varepsilon = \varepsilon^{\mathrm{e}} + \varepsilon^{\mathrm{p}} + \varepsilon^{\mathrm{v}} = \varepsilon^{\mathrm{e}} + \varepsilon^{\mathrm{vp}}. \tag{3.3}$$

The finite element structure computations under reverse cyclic loading are defined by strain, so for a given time (t) and from equation 3.3, the stress is calculated as:

$$\sigma(t) = E(\varepsilon(t) - \varepsilon^{\mathrm{vp}}(t)). \tag{3.4}$$

To determine the total strain in function of time, first the variable time needs to be converted to t_{cycle}, as described below:

$$t_{\mathrm{cycle}} = t - tc \left\lfloor \frac{t}{tc} \right\rfloor. \tag{3.5}$$

The variable t_c is the amount of time performed in the cyclic loading. For example, with a $tc = 20\,\text{s}$, in the first five seconds it will reach the maximum strain with traction. Five seconds further it reaches null strain. Then, it starts compressing until $t = 15\,\text{s}$. The cycle is complete after 20 s, where it reaches null strain. The total strain is calculated during this cycle by the following equation:

$$\varepsilon(t) = \begin{cases} \frac{4\varepsilon_{\max}}{t_c} t_{\text{cycle}}, & \text{if } t_{\text{cycle}} \leq \frac{t_c}{4} \\ 2\varepsilon_{\max} - \frac{4\varepsilon_{\max}}{t_c} t_{\text{cycle}}, & \text{if } \frac{t_c}{4} < t_{\text{cycle}} \leq \frac{t_c}{4} \\ 4\varepsilon_{\min} - \frac{4\varepsilon_{\min}}{t_c} t_{\text{cycle}}, & \text{if } \frac{3t_c}{4} < t_{\text{cycle}} \leq t_c \end{cases} \tag{3.6}$$

The time of cycle used during this research is $t_c = 20\,\text{s}$. An example for the previous described cyclic strain, with a range of ± 0.036 with four cycles is demonstrated in figure 3.1. In this example, the $\varepsilon_{\max} = 0.036$ and $\varepsilon_{\min} = -0.036$.

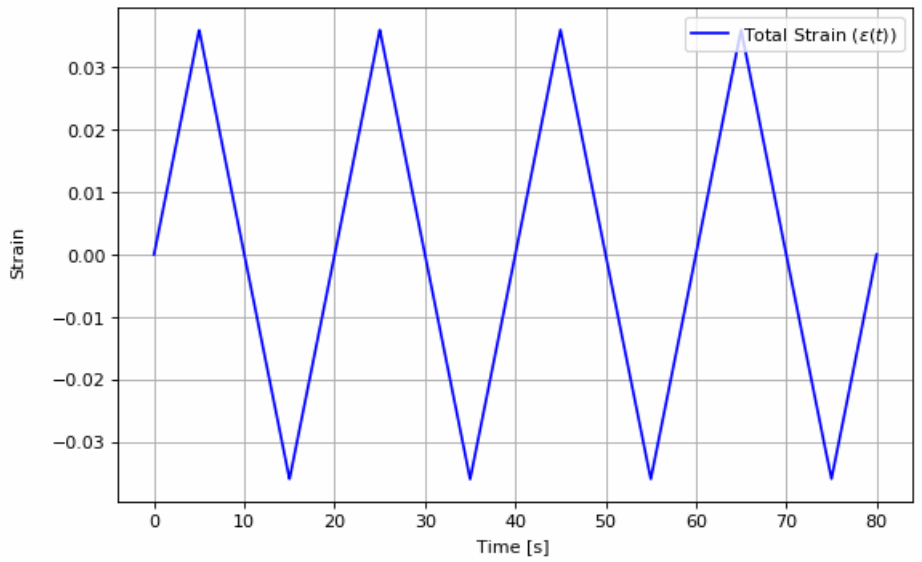


Figure 3.1: Total strain with a cyclic strain range of ± 0.036 during four cycles.

The viscoplastic component of the equation 3.4 is calculated by a differential equation that depends on the previous state of kinematic hardening (back stress) and isotropic hardening (drag stress) in order of time. These states are defined by differential equations and the properties of a specific material. The parameters used are presented in table 3.1.

The derivative of the viscoplastic ($\dot{\varepsilon}^{\text{vp}}$) component, as well as the back ($\dot{\chi}$) and drag stress ($\dot{R}$) rates in order to time, are defined by the following equations:

$$\dot{\varepsilon}^{\text{vp}} = \left(\frac{|\sigma - \chi| - R}{K} \right)^n sgn(\sigma - \chi), \tag{3.7a}$$

$$\dot{\chi} = H \cdot \dot{\varepsilon}^{\text{vp}} - D \cdot \chi |\dot{\varepsilon}^{\text{vp}}|, \tag{3.7b}$$

$$\dot{R} = h \cdot |\dot{\varepsilon}^{\text{vp}}| - d \cdot R |\dot{\varepsilon}^{\text{vp}}|. \tag{3.7c}$$

The relations between the differential equations presented in 3.7 can be visually represented by figure 3.2. This scheme represents the relations that the neural network have to mimic.

Table 3.1: Material parameters for the chaboche's viscoplasticity model in 1D.

Parameters	Values
E [MPa]	5000
K [MPa·$s^{1/n}$]	50
n [-]	3
H [MPa]	5000
h [MPa]	10
D [-]	300
d [-]	0.6

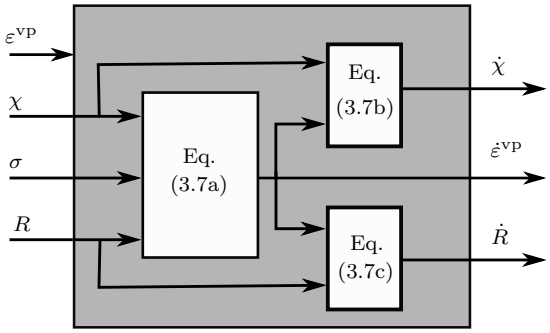


Figure 3.2: Scheme of equations from Chaboche's model in 1D for the machine learning model to learn.

3.1.1.1 Numerical Implementation

To solve the system of ordinary differential equations, `odeint` function from *Scipy*[1] ecosystem is used. This function needs as input the initial state of the system, a function which returns the derivatives for a given time, and a sequence of time points. A more detailed documentation can be found in [36]. The code used is shown in C.1 from appendix C. The algorithm used can be seen in algorithm 1.

The initial conditions are given by:

$$\varepsilon^{\text{vp}}|_{t=0} = \varepsilon_0^{\text{vp}} = 0; \tag{3.8a}$$

$$\chi|_{t=0} = \chi_0 = 0; \tag{3.8b}$$

$$R|_{t=0} = R_0 = 50. \tag{3.8c}$$

[1]Scipy ecosystem is an open-source Python library used for scientific computing and technical computing.

Algorithm 1: Function that returns viscoplastic strain, back and drag stress rate values at requested state.

Input: Initial conditions of the differential states at a given time t.

Output: Derivatives with respect to time: $\dot{\varepsilon}^{\text{vp}}, \dot{\chi}, \dot{R}$.

```
Calculate Total Stress
if elastic state then
    ε̇^vp = χ̇ = Ṙ = 0
    return null Derivatives;
else
    Calculate viscoplastic strain and state variables rates (equation 3.7)
    return Derivatives;
end
```

3.1.1.2 Virtually experimental results

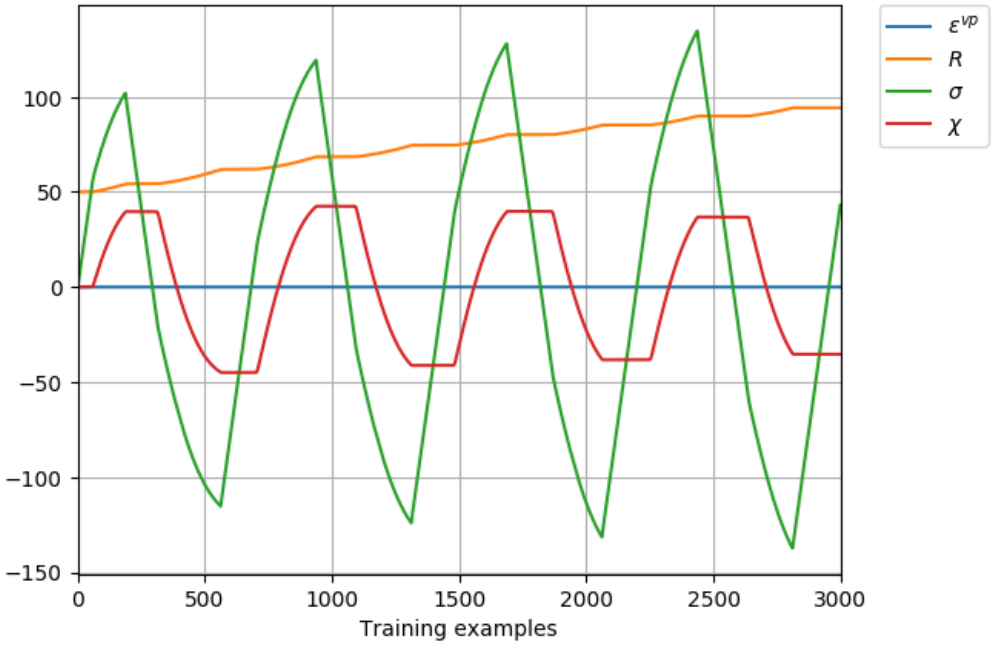
After solving the system of differential equations, the work carried out by the authors in the paper [4] was successfully reproduced. To train a neural network with the data obtained in this virtually experimental test, it is necessary to save the differential state (σ, ε^{vp}, χ and R) at each point of time. This result in the following matrix X, where the variable m corresponds to the number of time points, which is interpreted as number of training examples:

$$X_{(4,m)} = \begin{bmatrix} \sigma_{(1)} & \varepsilon^{\text{vp}}_{(1)} & \chi_{(1)} & R_{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{(m)} & \varepsilon^{\text{vp}}_{(m)} & \chi_{(m)} & R_{(m)} \end{bmatrix}. \tag{3.9}$$

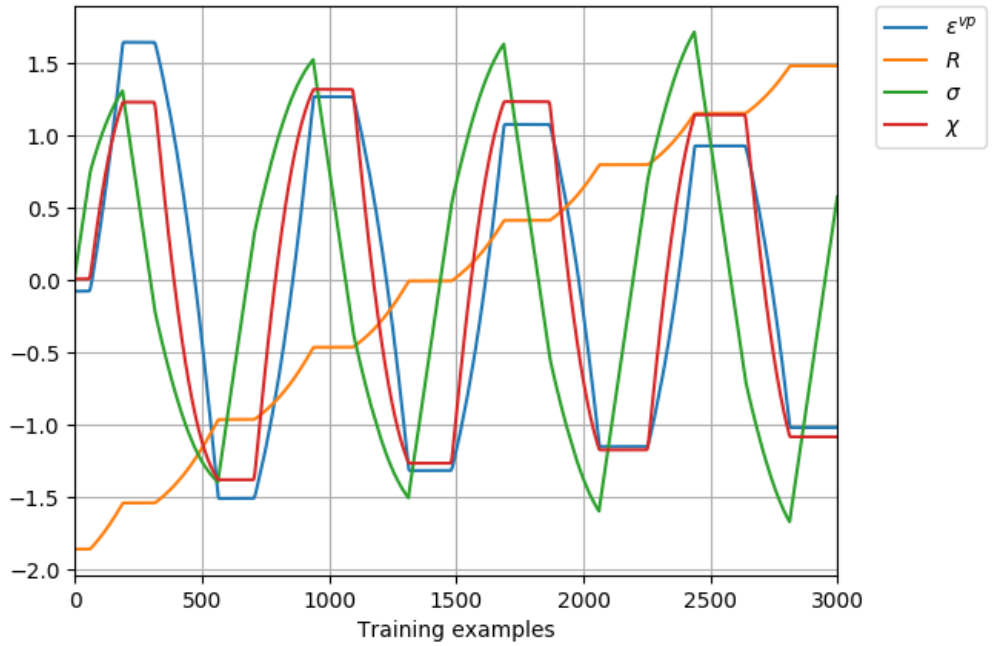
It is also necessary to save the corresponding derivatives for each differential state. For that, it is built the following target matrix y:

$$y_{(3,m)} = \begin{bmatrix} \dot{\varepsilon}^{\text{vp}}_{(1)} & \dot{\chi}_{(1)} & \dot{R}_{(1)} \\ \vdots & \vdots & \vdots \\ \dot{\varepsilon}^{\text{vp}}_{(m)} & \dot{\chi}_{(m)} & \dot{R}_{(m)} \end{bmatrix}. \tag{3.10}$$

The dataset is generated with $m = 3000$ and a cyclic strain range of ± 0.036, as shown in figure 3.1. To use it for training it is necessary to transform the X and y matrices. For this task it's used the `transform` method (see section 2.3.2). Figures 3.3 and 3.4 illustrates the input and the correspondent output values before and after the transformation, respectively.

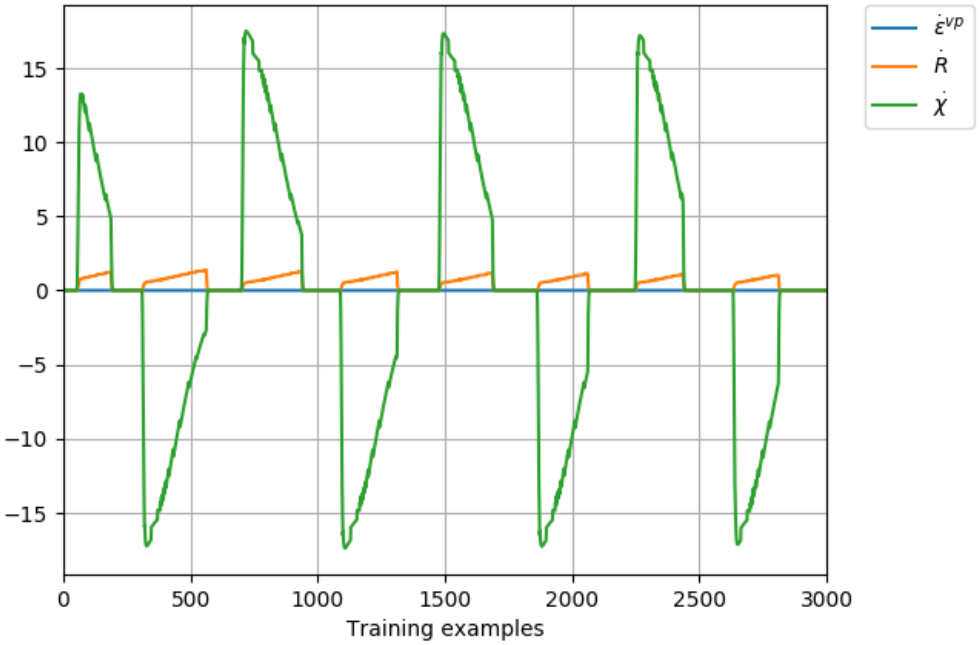


(a) Before transformation.

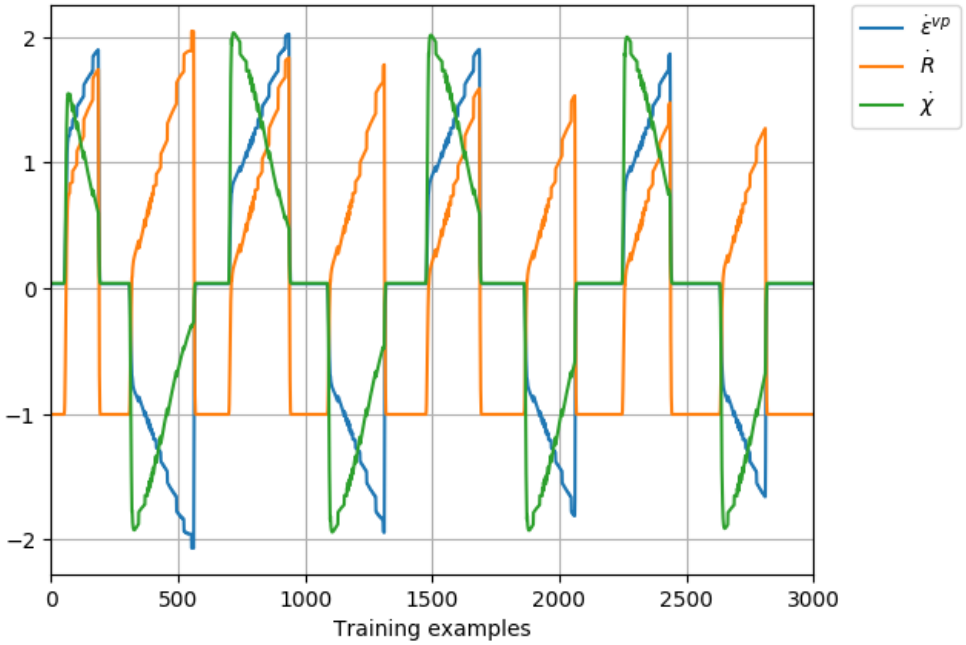


(b) After transformation.

Figure 3.3: Input values from the generated dataset in 1D before and after the transformation.



(a) Before transformation.



(b) After transformation.

Figure 3.4: Output values from the generated dataset in 1D before and after the transformation.

3.1.2 Chaboche Constitutive Model in 2D

The objective of this section is to compute an explicit viscoplastic material model in 2D. In order to simplify this computation, a plane stress condition is implemented. Under plane-stress conditions the following tensor:

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_{yy} & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_{zz} \end{bmatrix} \tag{3.11}$$

is simplified by the values establish below:

$$\tau_{xz} = \tau_{zx} = \tau_{yz} = \tau_{zy} = \sigma_{zz} = 0. \tag{3.12}$$

So in this case, Hooke's law takes the form represented on the matrix:

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \delta_{xy} \end{bmatrix} = \frac{1}{E} \begin{bmatrix} 1 & -\nu & 0 \\ -\nu & 1 & 0 \\ 0 & 0 & 2+2\nu \end{bmatrix} \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \tau_{xy} \end{bmatrix}. \tag{3.13}$$

Like in the previous section, the finite element structure computations under reverse cyclic loading are defined by strain. So, the inverse relation is usually written in the reduced form:

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \tau_{xy} \end{bmatrix} = \frac{E}{1-v^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \delta_{xy} \end{bmatrix}. \tag{3.14}$$

Although the stress tensor depends on the static material properties (Young's modulus and poisson coefficient), it also depends on the strain tensor. This vector is not the total strain, but the difference between the total and viscoplastic strain. In resemblance with the method implemented for the determination of total strain on the previous section, it is calculated through equations 3.5 and 3.6 for the 2D conditions:

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^t - \boldsymbol{\varepsilon}^{\mathrm{vp}}. \tag{3.15}$$

The viscoplastic strain rate tensor, $\dot{\boldsymbol{\varepsilon}}^{\mathrm{vp}}$, of the Chaboche model can be written as:

$$\dot{\boldsymbol{\varepsilon}}^{\mathrm{vp}} = \frac{3}{2}\dot{p}\frac{\boldsymbol{\sigma'} - \boldsymbol{\chi'}}{J(\boldsymbol{\sigma'} - \boldsymbol{\chi'})}. \tag{3.16}$$

The plastic strain rate, $\dot{p}$, has the following form:

$$\dot{p} = \left\langle \frac{J(\boldsymbol{\sigma'} - \boldsymbol{\chi'}) - R - k}{K} \right\rangle^n, \tag{3.17}$$

where the angle brackets represent the McCauley bracket, which is defined by:

$$\langle x \rangle = \frac{1}{2}(x + |x|). \tag{3.18}$$

Both inelastic and plastic strain rates, use the deviatoric back stress and the total deviatoric stress. But to calculate the deviatoric components, first it must be calculated the hydrostatic stress for both stresses. The Hydrostatic stress is the average of the three normal stress components of any stress tensor:

$$\sigma_{ii}^{hyd} = \frac{1}{3}(\sigma_{xx} + \sigma_{yy} + \sigma_{zz}), \tag{3.19}$$

$$\chi_{ii}^{hyd} = \frac{1}{3}(\chi_{xx} + \chi_{yy} + \chi_{zz}). \tag{3.20}$$

However, under plane stress conditions, the hydrostatic stress is the average of the two normal stress components, and the shear on hydrostatic stress is zero.

$$\boldsymbol{\sigma^{hyd}} = \begin{bmatrix} \frac{1}{2}(\sigma_{xx} + \sigma_{yy}) \\ \frac{1}{2}(\sigma_{xx} + \sigma_{yy}) \\ 0 \end{bmatrix}, \tag{3.21}$$

$$\boldsymbol{\chi^{hyd}} = \begin{bmatrix} \frac{1}{2}(\chi_{xx} + \chi_{yy}) \\ \frac{1}{2}(\chi_{xx} + \chi_{yy}) \\ 0 \end{bmatrix}. \tag{3.22}$$

The back and total deviatoric stresses are calculated by the following form:

$$\boldsymbol{\sigma'} = \boldsymbol{\sigma} - \boldsymbol{\sigma^{hyd}}, \tag{3.23}$$

$$\boldsymbol{\chi'} = \boldsymbol{\chi} - \boldsymbol{\chi^{hyd}}. \tag{3.24}$$

The J invariant is defined by the following equation, where the apostrophe between the parenthesis means to transpose the matrix:

$$J(\boldsymbol{\sigma'} - \boldsymbol{\chi'}) = \sqrt{\frac{3}{2}(\boldsymbol{\sigma'} - \boldsymbol{\chi'})'(\boldsymbol{\sigma'} - \boldsymbol{\chi'})}. \tag{3.25}$$

The evolution of the kinematic hardening rate (back stress rate), is defined by:

$$\dot{\boldsymbol{\chi}} = \frac{3}{2}a\dot{\boldsymbol{\varepsilon}}^{\mathrm{vp}} - c\boldsymbol{\chi}\dot{p}. \tag{3.26}$$

The isotropic hardening rate (drag stress rate) is calculated by the following equation:

$$\dot{R} = b(R_1 - R)\dot{p}. \tag{3.27}$$

The states of the previous differential equations depend also on material properties (E, ν, R_1, k, K, a, b, c, n). These parameters are presented on table 3.2.

Table 3.2: Material parameters for the Chaboche's viscoplasticity model in 2D.

Parameters	Values
E [MPa]	5000
ν[-]	0.3
R_1 [MPa]	500
k [MPa]	0
K[MPa $\cdot s^{1/n}$]	50
a [MPa]	7500
b [-]	0.6
c [-]	100
n [-]	3

The relations between the equations 3.24, 3.23, 3.25, 3.27, 3.17, 3.16 and 3.26, can be visually represented in figure 3.5. This scheme represents the constitutive description of the Chaboche's viscoplasticity model. Therefore, figure 3.5 summarizes the equations which the neural network will learn.

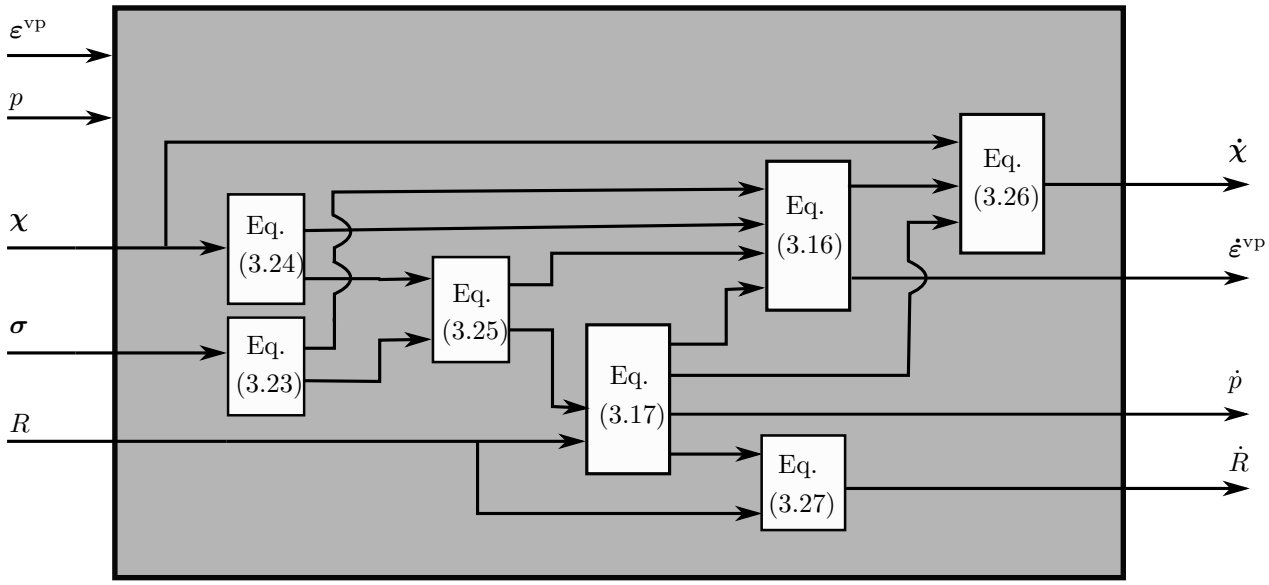


Figure 3.5: Scheme of equations from Chaboche's model in 2D for the machine learning model to learn.

3.1.2.1 Numerical Implementation

In resemblance to the numerical implementation described in 1D, the system of ordinary differential equations are solved using `odeint` function from *Scipy*[2] ecosystem. This function needs as input the initial state of the system, a function which returns the derivatives for a given time, and a sequence of time points. A more detailed documentation can be found in [36]. The code used is shown in C.2 from appendix C. The algorithm used can be seen in algorithm 2.

The initial conditions are given by:

$$\boldsymbol{\varepsilon}^{vp}|_{t=0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}; \tag{3.28a}$$

$$\boldsymbol{\chi}|_{t=0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}; \tag{3.28b}$$

$$R|_{t=0} = R_0 = 50; \tag{3.28c}$$

$$p|_{t=0} = p_0 = 0. \tag{3.28d}$$

Algorithm 2: Function that returns inelastic and plastic strain, back and drag stress rate values at a requested state.

Input: Initial conditions of the differential state and total strain at a given time t.

Output: Derivatives with respect to time: $\dot{\boldsymbol{\varepsilon}}^{\text{vp}}, \dot{\boldsymbol{\chi}}, \dot{R}, \dot{p}$.

Calculate:
- Total stress
- Back and total deviatoric stresses
- J invariant
- State

if *elastic state* **then**
- $\dot{\boldsymbol{\varepsilon}}^{\text{vp}} = \dot{\boldsymbol{\chi}} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
- $\dot{R} = \dot{p} = 0$
- return null Derivatives;

else
- Calculate viscoplastic and plastic strain and state variables rates
- return Derivatives;

end

[2]Scipy ecosystem is an open-source Python library used for scientific computing and technical computing.

3.1.2.2 Virtually experimental results

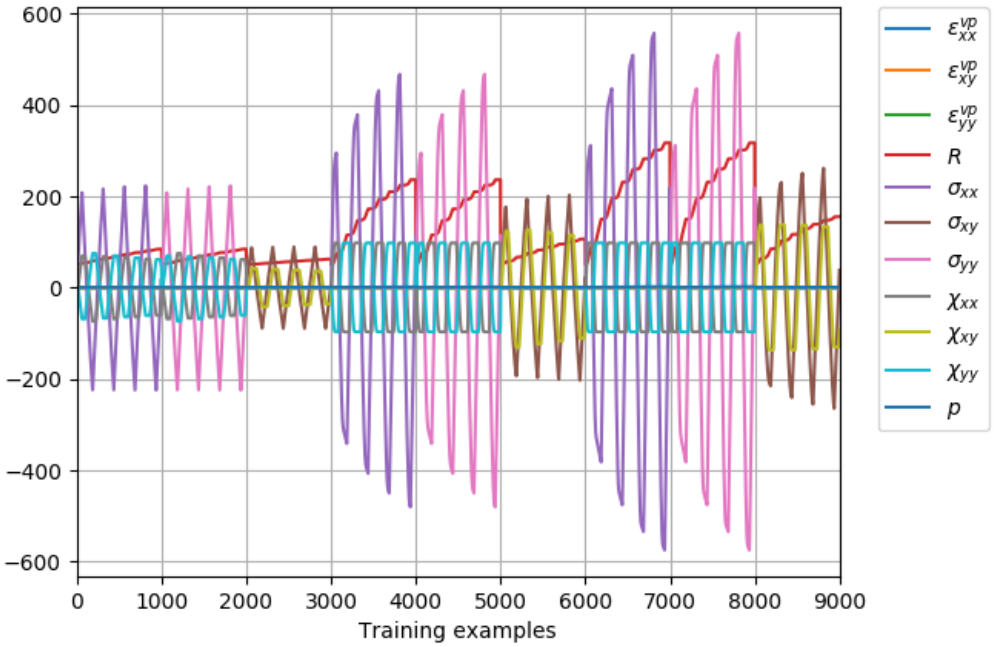
To train a neural network with the data obtained in this virtually experimental test, it is necessary to save the differential state ($\boldsymbol{\sigma}$, $\boldsymbol{\varepsilon}^{\text{vp}}$, $\boldsymbol{\chi}$, R and p) at each point of time. However, plane-stress conditions leads to three types of testing: the tensile tests along xx (T_{xx}), yy (T_{yy}) axis and simple shear (S_{xy}). In addiction, the tests can be performed under various cyclic strain ranges. This results in the following matrix X, where the variable m and n are the number of time points and the correspondent cyclic strain range, respectively:

$$X_{(11,m\times 3\times n)} = \begin{bmatrix} \boldsymbol{\sigma}_{(1,1,n)} & \boldsymbol{\varepsilon}^{\text{vp}}_{(1,1,n)} & \boldsymbol{\chi}_{(1,1,n)} & R_{(1,1,n)} & p_{(1,1,n)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{\sigma}_{(m,1,n)} & \boldsymbol{\varepsilon}^{\text{vp}}_{(m,1,n)} & \boldsymbol{\chi}_{(m,1,n)} & R_{(m,1,n)} & p_{(m,1,n)} \\ \boldsymbol{\sigma}_{(1,2,n)} & \boldsymbol{\varepsilon}^{\text{vp}}_{(1,2,n)} & \boldsymbol{\chi}_{(1,2,n)} & R_{(1,2,n)} & p_{(1,2,n)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{\sigma}_{(m,2,n)} & \boldsymbol{\varepsilon}^{\text{vp}}_{(m,2,n)} & \boldsymbol{\chi}_{(m,2,n)} & R_{(m,2,n)} & p_{(m,2,n)} \\ \boldsymbol{\sigma}_{(1,3,n)} & \boldsymbol{\varepsilon}^{\text{vp}}_{(1,3,n)} & \boldsymbol{\chi}_{(1,3,n)} & R_{(1,3,n)} & p_{(1,3,n)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{\sigma}_{(m,3,n)} & \boldsymbol{\varepsilon}^{\text{vp}}_{(m,3,n)} & \boldsymbol{\chi}_{(m,3,n)} & R_{(m,3,n)} & p_{(m,3,n)} \end{bmatrix}. \tag{3.29}$$

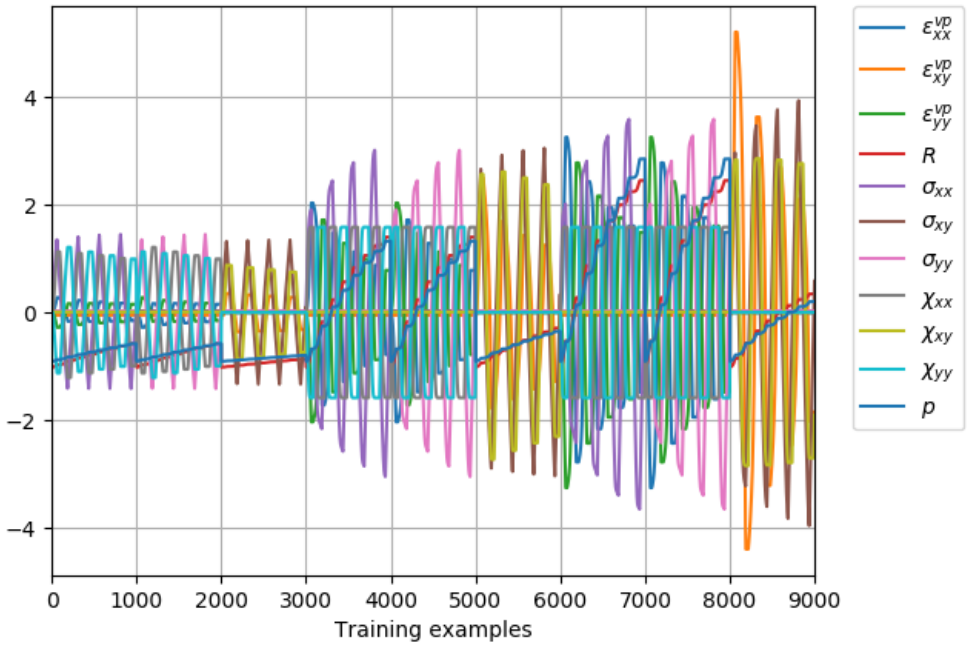
It is also necessary to save the corresponding derivatives for each differential state. For that, the following target matrix y is built with those derivatives:

$$y_{(8,m\times 3\times n)} = \begin{bmatrix} \dot{\boldsymbol{\varepsilon}}^{\text{vp}}_{(1,1,n)} & \dot{\boldsymbol{\chi}}_{(1,1,n)} & \dot{R}_{(1,1,n)} & \dot{p}_{(1,1,n)} \\ \vdots & \vdots & \vdots & \vdots \\ \dot{\boldsymbol{\varepsilon}}^{\text{vp}}_{(m,1,n)} & \dot{\boldsymbol{\chi}}_{(m,1,n)} & \dot{R}_{(m,1,n)} & \dot{p}_{(m,1,n)} \\ \dot{\boldsymbol{\varepsilon}}^{\text{vp}}_{(1,2,n)} & \dot{\boldsymbol{\chi}}_{(1,2,n)} & \dot{R}_{(1,2,n)} & \dot{p}_{(1,2,n)} \\ \vdots & \vdots & \vdots & \vdots \\ \dot{\boldsymbol{\varepsilon}}^{\text{vp}}_{(m,2,n)} & \dot{\boldsymbol{\chi}}_{(m,2,n)} & \dot{R}_{(m,2,n)} & \dot{p}_{(m,2,n)} \\ \dot{\boldsymbol{\varepsilon}}^{\text{vp}}_{(1,3,n)} & \dot{\boldsymbol{\chi}}_{(1,3,n)} & \dot{R}_{(1,3,n)} & \dot{p}_{(1,3,n)} \\ \vdots & \vdots & \vdots & \vdots \\ \dot{\boldsymbol{\varepsilon}}^{\text{vp}}_{(m,3,n)} & \dot{\boldsymbol{\chi}}_{(m,3,n)} & \dot{R}_{(m,3,n)} & \dot{p}_{(m,3,n)} \end{bmatrix}. \tag{3.30}$$

The dataset is created with 1000 data points for all virtually experimental tests, resulting in 9000 training examples. The mechanical displacements used to generate the virtually experimental tests are illustrated on figures A.1, A.2, A.3, A.4, A.5, A.6, A.7, A.8 and A.9 from appendix A. In resemblance to the transformation performed to the data for the 1D model, it is used the `transform` method (see section 2.3.2) to transform the X and y matrices. Figures 3.6 and 3.7 shows the input and the correspondent output values before and after the transformation, respectively.

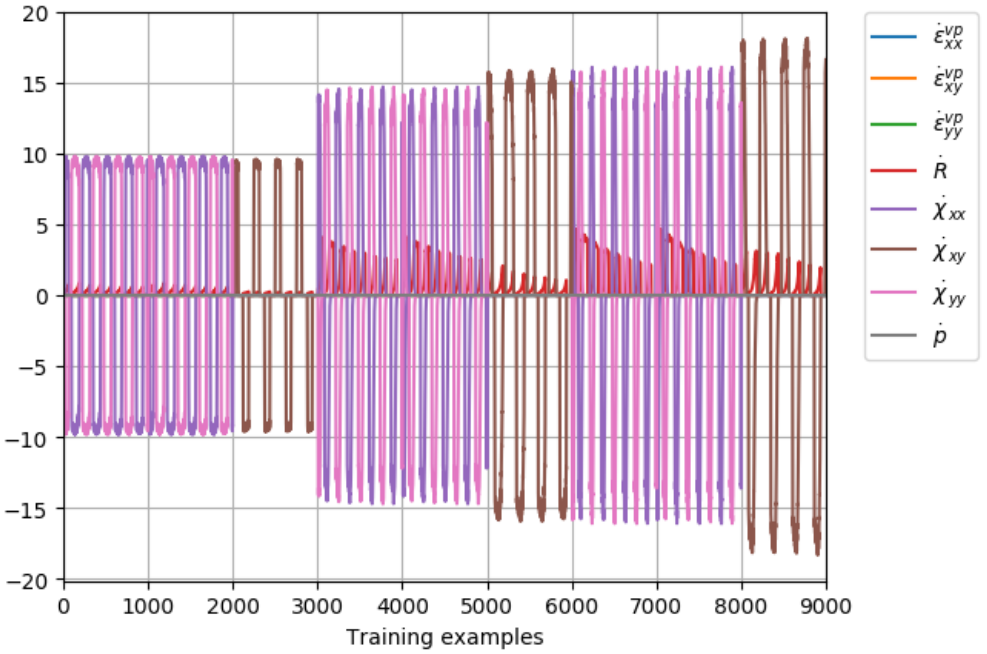


(a) Before transformation.

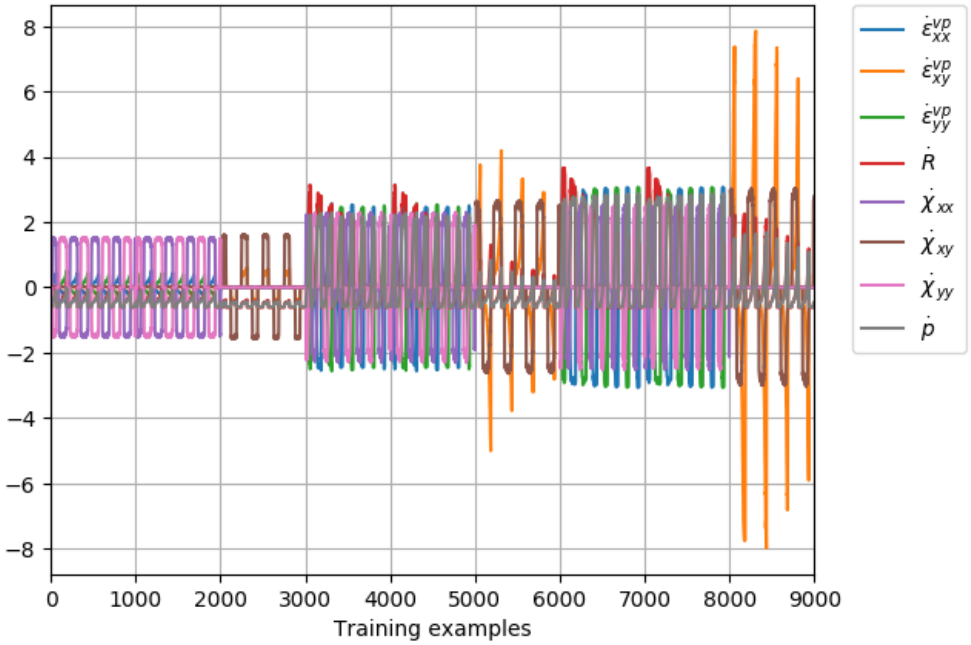


(b) After transformation.

Figure 3.6: Input values from the generated dataset in 2D before and after the transformation.



(a) Before transformation.



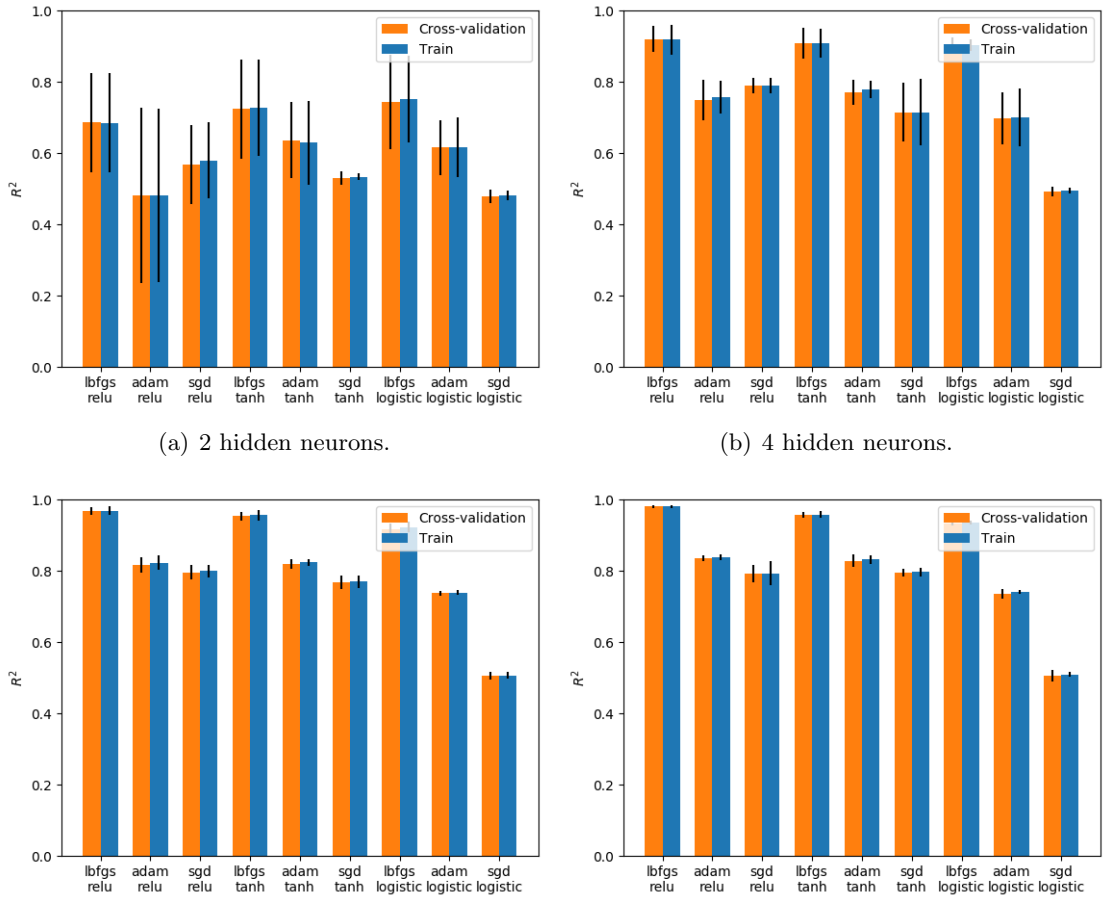
(b) After transformation

Figure 3.7: Output values from the generated dataset in 2D before and after the transformation.

3.2 Training on Generated Dataset

3.2.1 ANN Model for 1D

On a first step, the selection of the optimization algorithm and the activation function is performed. For this task, it's used the grid search function from Scikit-Learn using cross validation with 5 folds. This search makes a combination between three activation functions (logistic, tanh, relu), three optimization algorithms (lbfgs, sgd, adam) and four topologies (two, four, seven and ten hidden neurons) for one hidden layer. All estimators are trained with an automatic batch size=min(200,number of training examples), unless if the optimization algorithm is lbfgs. In this case the estimators will not use minibatch. The linear activation function is always used for the output layer and the learning rate is set constant with an initial value of 0.001.



(a) 2 hidden neurons.

(b) 4 hidden neurons.

(c) 7 hidden neurons.

(d) 10 hidden neurons.

Figure 3.8: Grid search results for the 1D training dataset using different optimization algorithms, activation functions and neural network sizes.

The results obtained by the grid search are illustrated on figure 3.8. It is possible to observe that `lbfgs` optimization algorithm produces the higher score regardless of the activation function or the number of hidden neurons used. Although, `relu` and `tanh` produce the best

scores as activation functions, it is the use of `lbfgs` and `relu` that overcome any combination of optimization algorithms and activation functions for this dataset.

The next step was to choose the neural network size. For this purpose, it was used `validation_curve` function from Scikit-Learn library with five folds ($k = 5$). It was tested for one hidden layer with a range of hidden neurons from 1 to twenty. Figure 3.9 presents the results. It can be conclude that 4 neurons is the optimal value, with $R^2 = 0.971$ for training score mean and $R^2 = 0.969$ for cross-validation score mean.

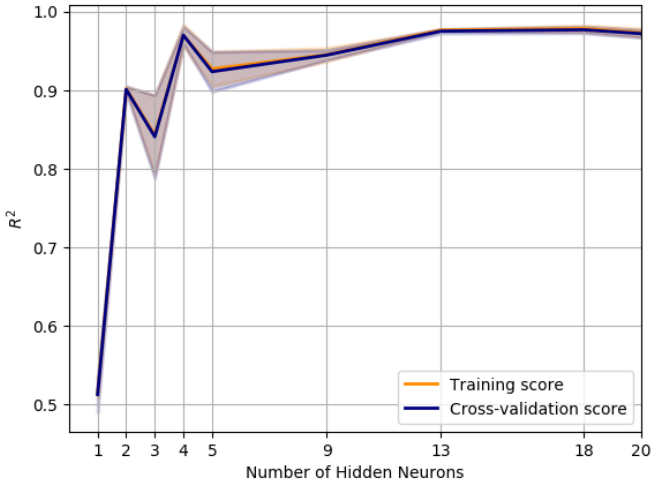


Figure 3.9: Validation curve of trained 1D model with a range of hidden neurons from one to twenty for one hidden layer.

In terms of regularization, the `validation_curve` was used to test model's performance using different values of regularization factors (λ), using a cross-validation with 5 folds. Figure 3.23 presents the results, which tells that the model has to be trained with $\gamma = 0.1$, in order to obtain a neural network model with the highest performance.

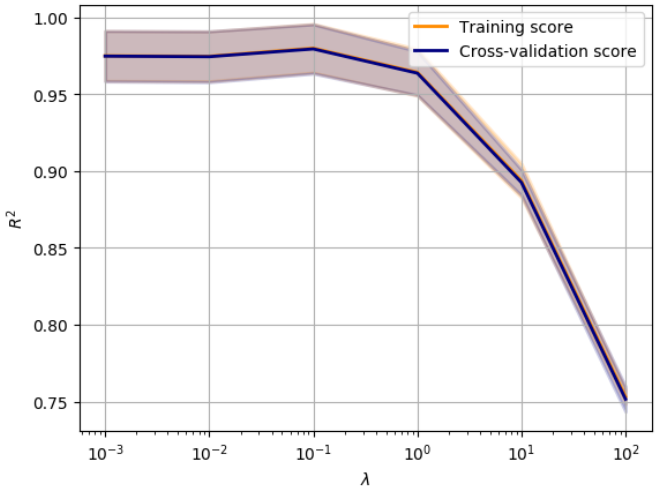


Figure 3.10: Validation curve of trained 1D model with a range of regularization factor values.

3.2.1.1 Final model

Regarding the variables which establish how the network is trained, the rectified linear unit (`relu`) was selected as activation function for the hidden layer. For the training solver, Limited-memory BFGS (`lbfgs`), an optimization algorithm in the family of quasi-Newton methods is selected. The learning rate was set to constant with an initial learning rate of 0.001. To penalty the weight updates, regularization was used with $\lambda = 0.1$, and the number of epochs was set to automatic. The neural network have an input layer with four nodes (total stress, viscoplastic strain, back and drag stress), only one hidden layer with 4 nodes and an output layer with three nodes (viscoplastic strain rate, back and drag stress rates). All of the previous described parameters are listed in table 3.3.

The dataset have a total of 3000 number of examples. The model, after fitting on training data, presented a R^2 of 97,28% and a MSE of 0.0333.

Table 3.3: Final model variables described for training, neural network's architecture, the score of predicting the material behavior and it's mean-square-error.

Activation Function	**Optimization Algorithm**	**Learning Rate**	**Number of Neurons**	$\boldsymbol{\lambda}$	**Number of Iterations**	$\boldsymbol{R^2}$	**MSE**
relu	lbfgs	constant	4	0.1	103	0.9728	0.0333

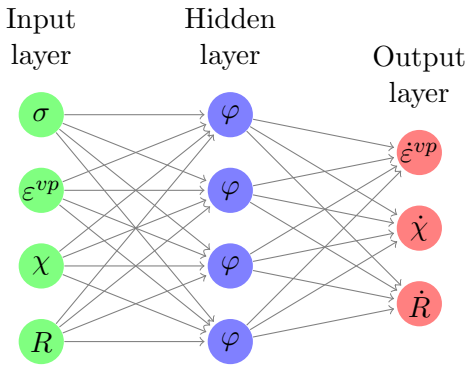


Figure 3.11: Representation of the final neural network's architecture for the 1D model. It is composed with an input layer with four nodes (total stress, viscoplastic strain, back and drag stress), one hidden layer with 4 nodes and an output layer with three nodes (viscoplastic strain rate, back and drag stress rates).

The error development of the training set until 103 training iterations is shown in figure 3.12. The error is approaching to zero, indicating that the neural network is learning the material model.

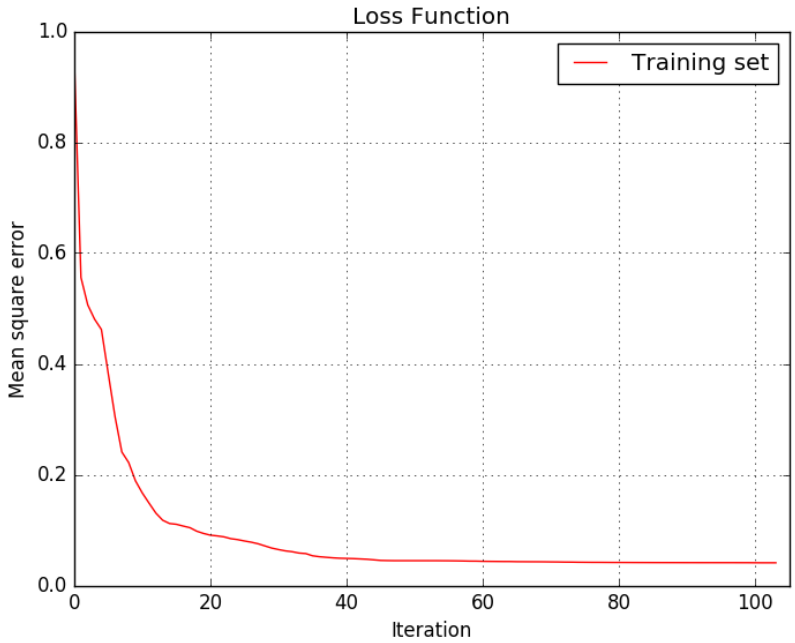
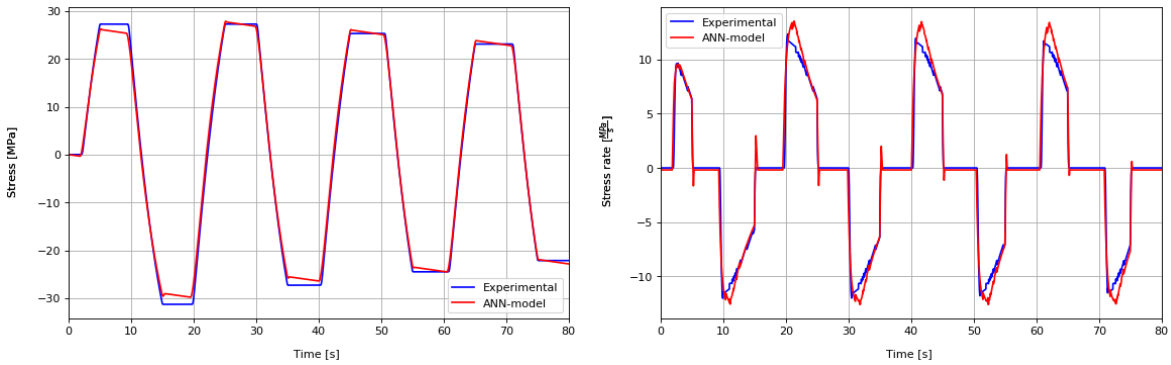


Figure 3.12: MSE function values from training set thought 103 iterations.

3.2.1.2 Validation

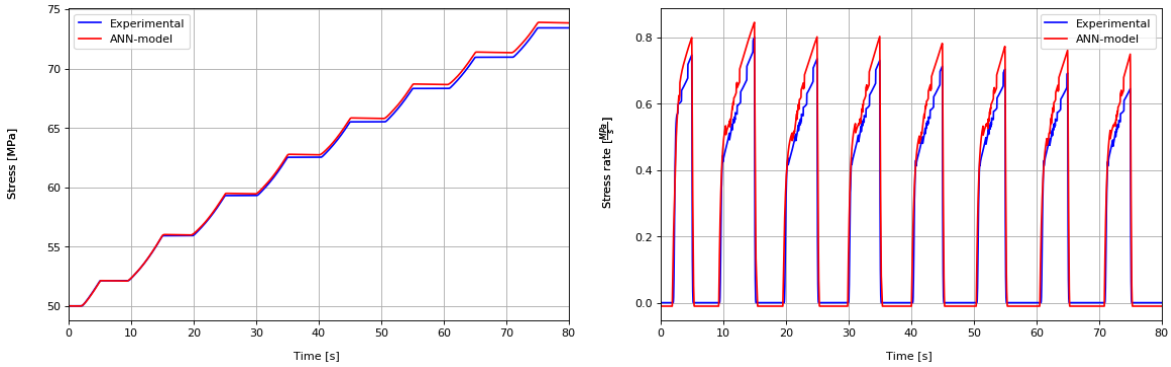
Now that the proposed model can reproduce the training data, the total strain needs to be changed in order to validate the trained neural network model to produce untrained curves. For this effect, the performance of the model with cyclic strains ranges of $\pm 0.025\,\%$, $\pm 0.040\,\%$ and $\pm 0.072\,\%$ were simulated and compared to the virtually experimental results. These values were selected in resemblance to the work done in [4].

The figures 3.13, 3.14 and 3.15 present the comparison between the experimental and ANN-model values of back stress, drag stress and viscoplastic strain respectively, together with their derivatives. These figures only represent the correspondent cyclic strain of $\pm 0.025\,\%$.



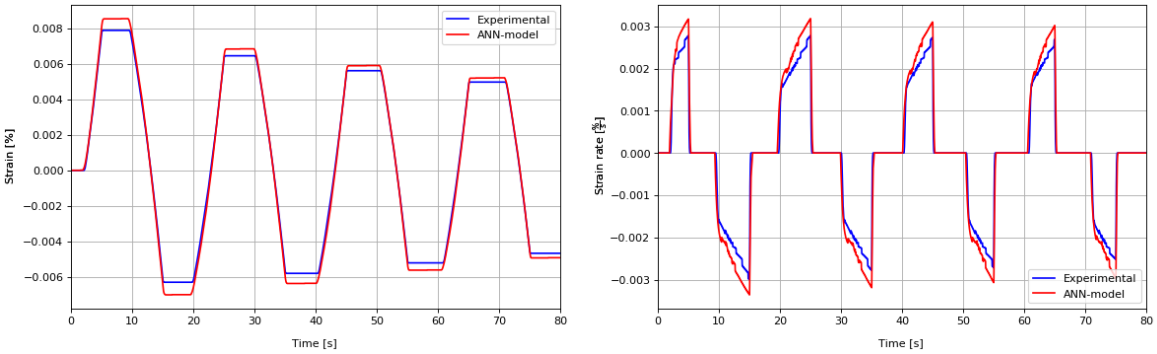
(a) Back stress. (b) Back stress rate.

Figure 3.13: Comparison between predicted and experimental values of back stress and it's rate over time with a cyclic strain of $\pm 0.025\,\%$.



(a) Drag stress. (b) Drag stress rate.

Figure 3.14: Comparison between predicted and experimental values of drag stress and it's rate over time with a cyclic strain of $\pm 0.025\,\%$.



(a) Inelastic strain.

(b) Inelastic strain rate.

Figure 3.15: Comparison between predicted and experimental values of viscoplastic strain, and it's rate over time with a cyclic strain of $\pm 0.025\,\%$.

Gathering predicted and experimental values of drag stress, back stress and total stress resulted in figure 3.16. This comparison shows that the ANN-model could produce these untrained curves with success, since the experimental and the predicted curves match in all three stresses. For example, at the maximum strain of the first cycle ($t = 5\,\mathrm{s}$), the total, back and drag stress and viscoplastic strain are 85.61 MPa, 27.05 MPa, 51.93MPa and $7.79\times10^{-3}\%$ respectively. Whereas the corresponding curve by the proposed model has a total stress of 83.60 MPa, a back stress of 25.86 MPa, a drag stress of 52.04 MPa and a viscoplastic strain of $8.17\times10^{-3}\%$. This relation and the percentage error are described in table 3.4.

The cyclic loading with a cyclic strain range of $\pm 0.025\,\%$ is represented in figure 3.17.

Table 3.4: Values calculated with the proposed model in comparison with the experimental results in 1D, as well as the percentage error between them. These values correspond at the maximum strain of the first cycle ($t = 5\,\mathrm{s}$) on a strain range of $\pm 0.025\,\%$.

	Total Stress [MPa]	**Back Stress [MPa]**	**Drag Stress [MPa]**	**Viscoplastic Strain [%]**
ANN-model	83.60	25.86	52.04	8.17×10^{-3}
Experimental	85.61	27.05	51.93	7.79×10^{-3}
Error [%]	2.35	4.34	0.211	4.88

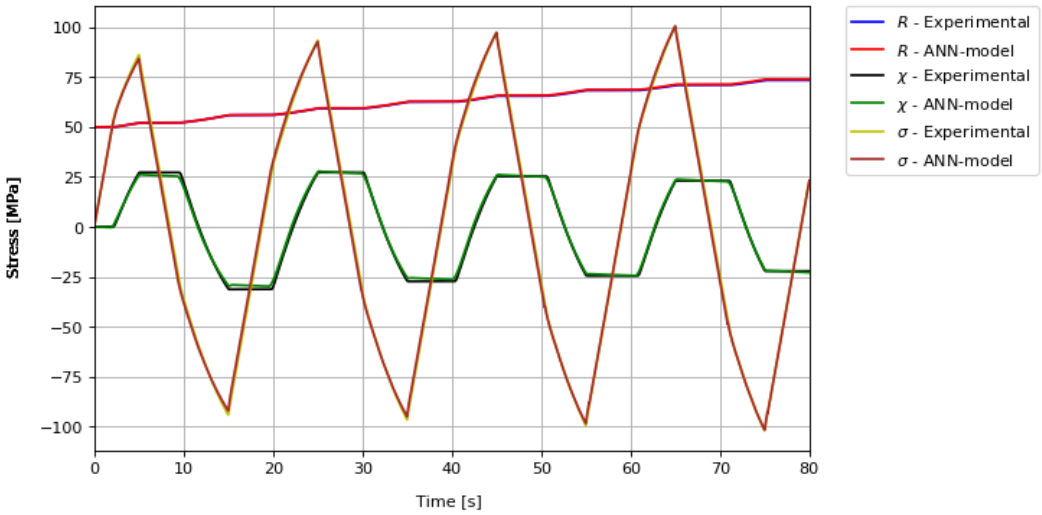


Figure 3.16: Comparison between predicted and experimental values of all stresses over time with a cyclic strain of $\pm 0.025\,\%$.

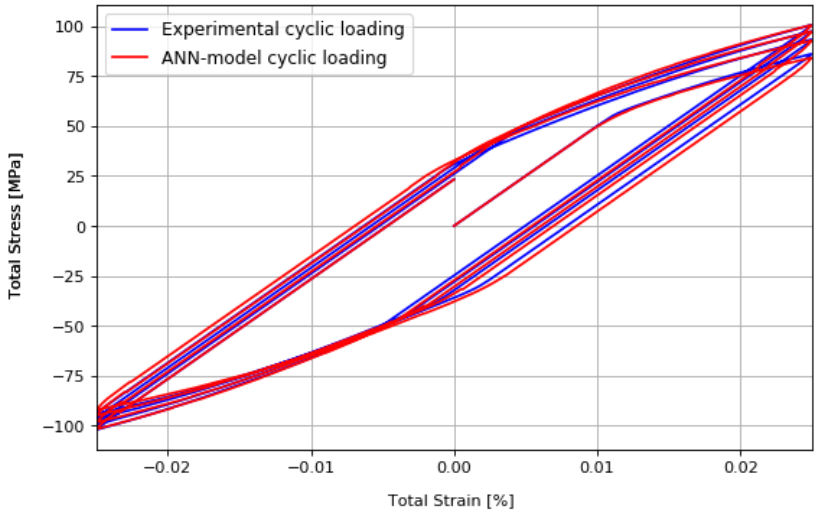


Figure 3.17: Comparison between predicted and experimental values of cyclic loading over time with a cyclic strain of $\pm 0.025\,\%$.

An identical material behaviour to the Chaboche's model is obtained with a cyclic strain range of $\pm 0.040\,\%$ as shown in figure 3.18. It should be noticed that the range of this test exceeds the one of the training data. At the maximum strain of the first cycle ($t = 5\,\mathrm{s}$), the total stress is 105.33 MPa, whereas the corresponding curve by the proposed model has a total stress of 109.62 MPa. In table 3.5 is possible to visualize the comparison of all stresses and viscoplastic strain between the experimental results and the predictions of the proposed model at the first maximum strain, as well as the percentage error.

Table 3.5: Values calculated with the proposed model in comparison with the experimental results in 1D, as well as the percentage error between them. These values correspond at the maximum strain of the first cycle ($t = 5\,\mathrm{s}$) on a strain range of $\pm 0.040\,\%$.

	Total Stress [MPa]	**Back Stress [MPa]**	**Drag Stress [MPa]**	**Viscoplastic Strain [%]**
ANN-model	109.62	43.32	54.52	1.79×10^{-2}
Experimental	105.33	42.37	55.04	1.88×10^{-2}
Error [%]	4.07	2.24	0.94	4.79

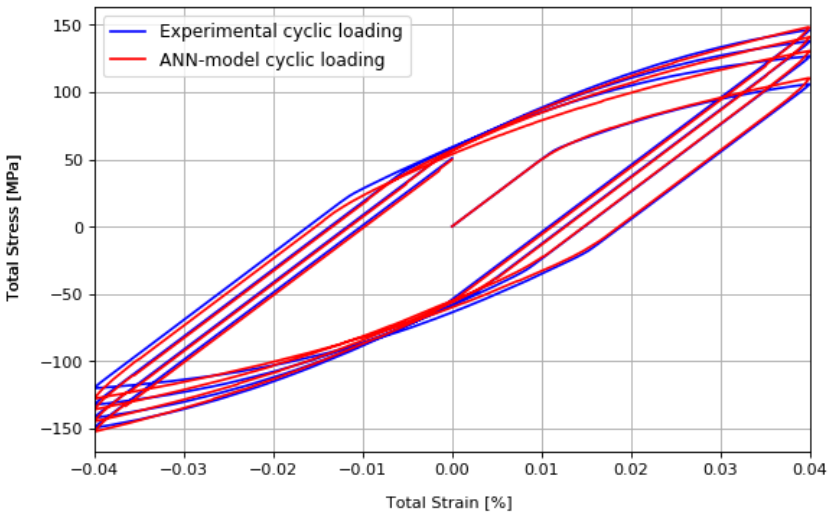


Figure 3.18: Comparison between predicted and experimental values of cyclic loading over time with a cyclic strain of $\pm 0.040\,\%$.

The curves of the proposed model with a cyclic strain range of $\pm 0.072\,\%$ are far off the experimental curves as illustrated in figure 3.19. At the maximum strain of the first cycle ($t = 5\,\mathrm{s}$), the total stress is 121.73 MPa, whereas the corresponding curve by the proposed model has a total stress of 162.55 MPa. Table 3.6 presents the percentage error among the predictions made by the neural network with the experimental results at the first maximum strain.

Table 3.6: Values calculated with the proposed model in comparison with the experimental results in 1D, as well as the percentage error between them. These values correspond at the maximum strain of the first cycle ($t = 5\,\mathrm{s}$) on a strain range of $\pm 0.072\,\%$.

	Total Stress [MPa]	**Back Stress [MPa]**	**Drag Stress [MPa]**	**Viscoplastic Strain [%]**
ANN-model	162.55	77.44	59.94	3.91×10^{-2}
Experimental	121.73	49.56	62.62	4.74×10^{-2}
Error [%]	33.53	56.25	4.28	17.51

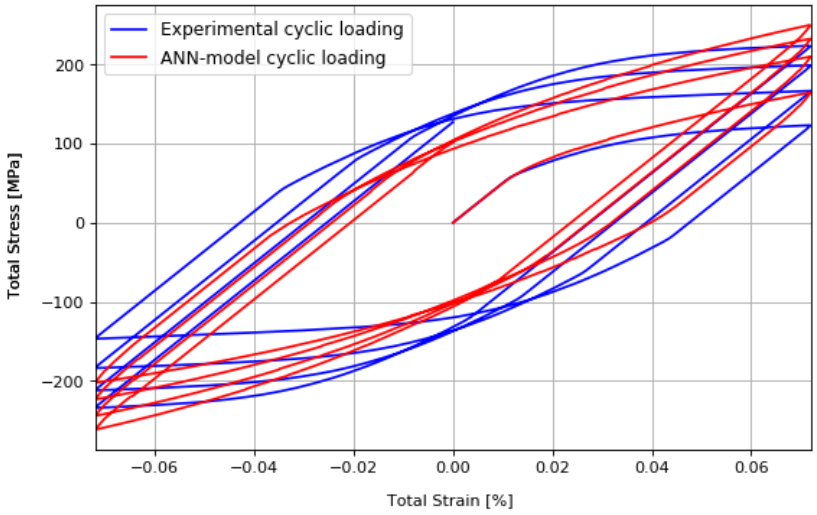


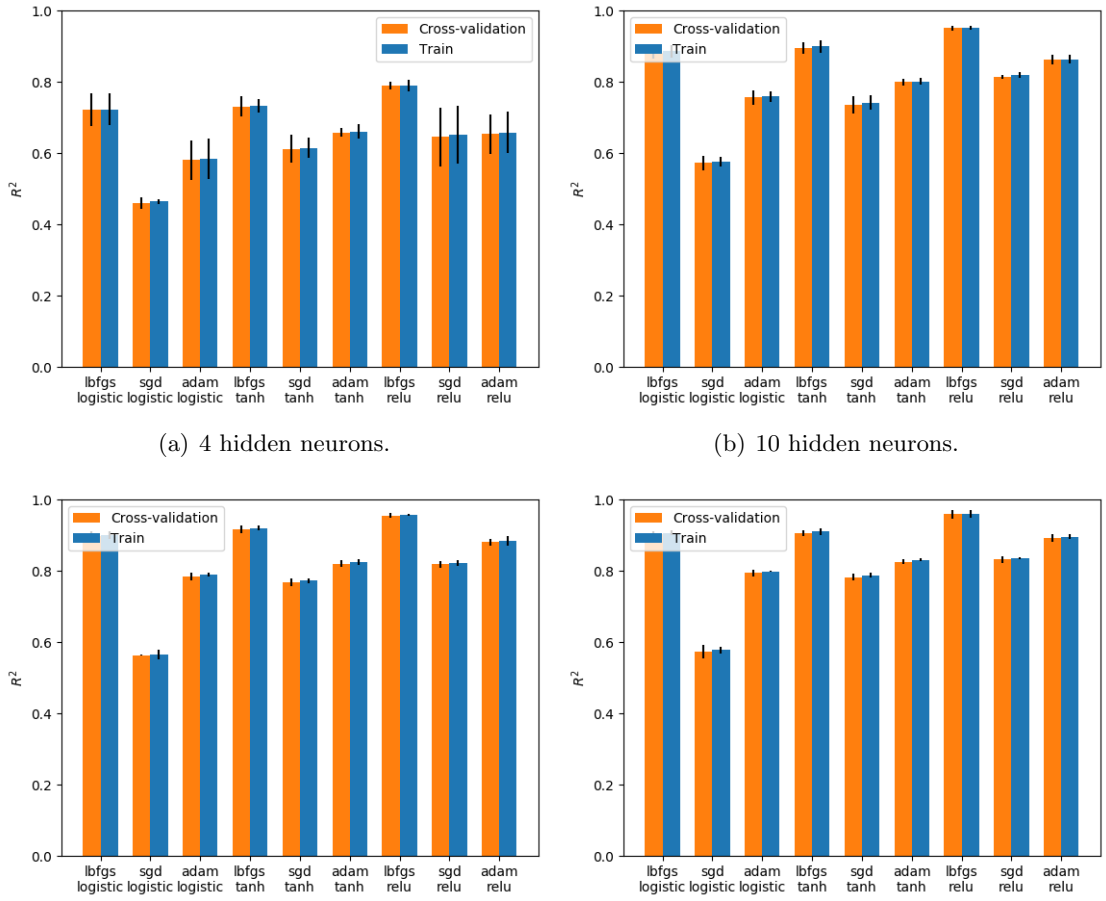
Figure 3.19: Comparison between predicted and experimental values of cyclic loading over time with a cyclic strain of $\pm 0.072\,\%$.

In conclusion, the trained neural network has a good performance using a cyclic strain range of $0.025\,\%$ and $0.040\,\%$. Although, when using a cyclic strain of $0.072\,\%$, it fails to predict the stress and the state variables. This is due to the fact that the cyclic strain used is two times higher than the one used for training.

3.2.2 ANN Model for 2D

The previous dataset for the 1D viscoplasticity model has four features (σ, ε^{vp}, χ and R) and three outputs ($\dot{\varepsilon}^{\text{vp}}$, $\dot{\chi}$ and $\dot{R}$). In comparison with the 2D viscoplastic model, the complexity of the data is smaller in the 1D model. This is because the dataset used to mimic the 2D viscoplastic model has eleven features ($\boldsymbol{\sigma}$, $\boldsymbol{\varepsilon}^{\text{vp}}$, $\boldsymbol{\chi}$, R and p) and eight outputs ($\dot{\boldsymbol{\varepsilon}}^{\text{vp}}$, $\dot{\boldsymbol{\chi}}$, $\dot{R}$ and $\dot{p}$). In addiction, under plane-stress conditions there are three different tensile tests, which add complexity to the dataset.

To overcome the problem of complexity, the dataset was created with three cyclic strain ranges of ± 0.05, ± 0.12 and ± 0.16 for each different trials, containing 1000 data points each, therefore providing more information about the material behavior. The 1D model was only trained with one cyclic strain range. In addiction, the neural network size is increased.



(a) 4 hidden neurons.

(b) 10 hidden neurons.

(c) 16 hidden neurons.

(d) 22 hidden neurons.

Figure 3.20: Grid search results for the ANN-model for the 2D experimental dataset using different optimization algorithms, activation functions and neural network sizes.

In resemblance with the selection of the optimization algorithm and the activation function performed on the training process for the 1D model, it's used the grid search function, using cross-validation with 5 folds. A combination between three activation functions (logistic, tanh, relu), three optimization algorithms (lbfgs, sgd, adam) and four topologies (four, ten, sixteen and twenty two hidden neurons) for one hidden layer is conducted for this search. The estimators are trained with an automatic batch size=min(200,number of training examples), unless if the optimization algorithm is lbfgs. In this case the estimator will not use minibatch. The linear activation function is always used for the output layer and the learning rate is set constant with an initial value of 0.001.

The results obtained by the grid search are illustrated on figure 3.20. It is possible to observe that similar results are obtained when comparing this search with the one performed on the 1D model. The combination of `lbfgs` optimization algorithm and `relu` as activation function produce the highest score, regardless of the number of hidden neurons.

The next step was to choose the neural network size. For this purpose, it was used the `validation_curve` function from Scikit-Learn library with five folds ($k = 5$). It was tested for one hidden layer with a range of hidden neurons from four to twenty. Figure 3.21 presents the results. It can be conclude that 16 neurons is the optimal value, with $R^2 = 0.965$ for training score mean and $R^2 = 0.964$ for cross-validation score mean.

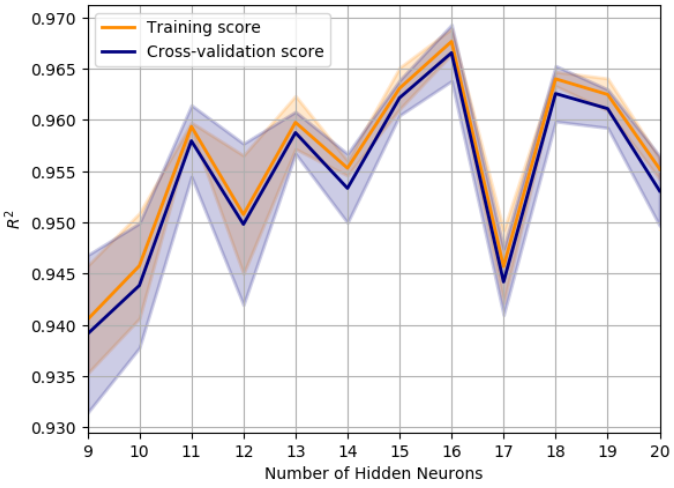


Figure 3.21: Validation curve of trained ANN-model with a range of hidden neurons from nine to twenty for one hidden layer.

Then, the `learning_curve` function from Scikit-Learn was used to determine if the model could perform better with more training examples, using a 5 fold cross-validation and with one hidden layer with 16 hidden nodes. The results are illustrated in figure 3.22. This figure shows that adding more training examples is not likely to help the model, due to the fact the training and cross validation R^2 mean values converge and stay almost the same after 6000 training examples.

In terms of regularization, the `validation_curve` was used to test model's performance using different values of λ, using a cross-validation with 5 folds. Figure 3.23 presents the results, which tells that the model has to be trained with a regularization factor lower than 0.01 in order to preserve the performance.

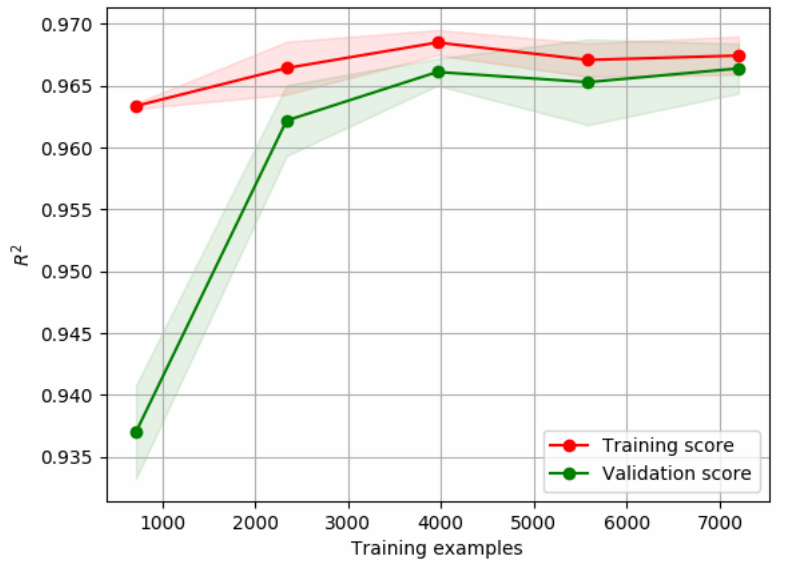


Figure 3.22: Learning curve of ANN-model for the 2D experimental dataset.

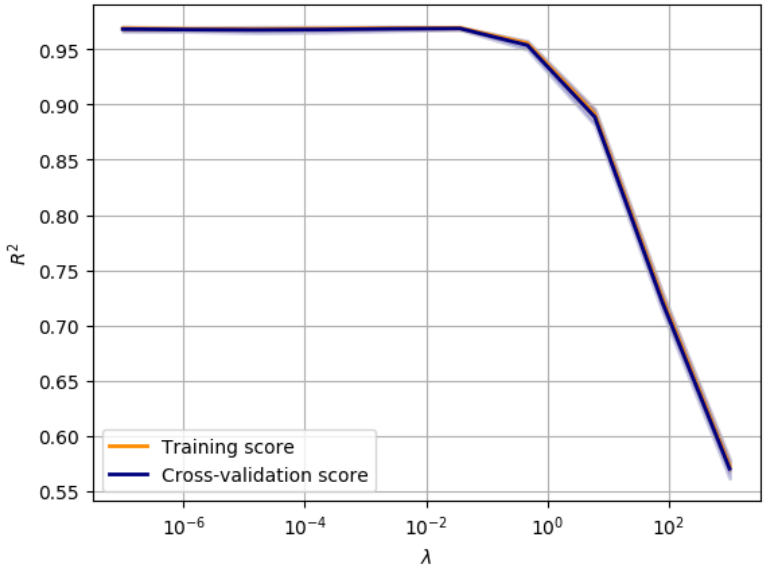


Figure 3.23: Validation curve of ANN-model with a range of λ.

3.2.2.1 Final model

Regarding the variables which establish how the network is trained, the rectified linear unit (relu) was selected as activation function for the hidden layer. The used training solver was the Limited-memory BFGS (lbfgs), an optimization algorithm in the family of quasi-Newton methods. The learning rate was set to `constant` with an initial learning rate of 0.001. The weights updates were penalized, with a regularization factor $\lambda = 0.001$, and the number of epochs was set to automatic. The neural network has an input layer with eleven nodes (total stress tensor, viscoplastic strain tensor, plastic strain, back stress tensor and drag stress), one hidden layer with sixteen nodes and an output layer with eight nodes (viscoplastic strain rate tensor, back stress rate tensor, plastic strain and drag stress rates). All of the previous described parameters are listed in table 3.7.

The ANN-model, after fitting on training data, presented a score (R^2) of 96,9% and a mean square error (MSE) of 0.0153.

Table 3.7: Final model variables described for training, neural network's architecture, the score of predicting the material behavior and it's mean-square-error.

Activation Function	**Optimization Algorithm**	**Learning Rate**	**Number of Neurons**	$\boldsymbol{\lambda}$	**Number of Iterations**	$\boldsymbol{R^2}$	**MSE**
relu	lbfgs	constant	16	0.001	190	0.969	0.0153

The evolution of the error during the training through 190 iterations is shown in figure 3.24. The error reaches almost zero, indicating that the neural network is learning the material model.

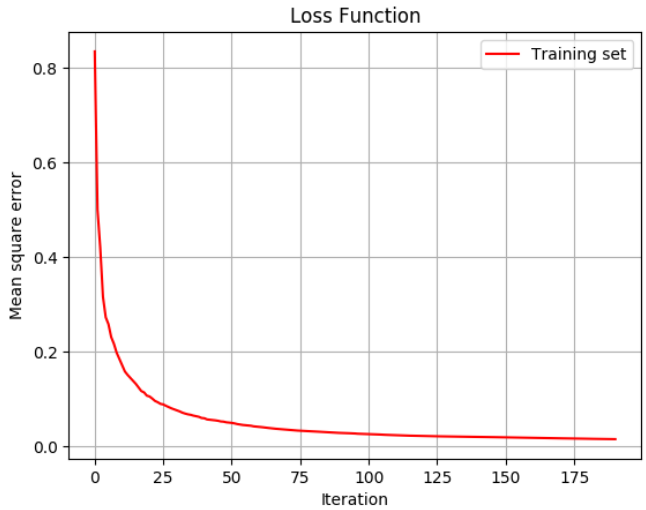


Figure 3.24: MSE function values from training set throughout 190 iterations.

3.2.2.2 Validation

The performance of the model with cyclic strain ranges of ± 0.11 , ± 0.14 and ± 0.18, for each test, were simulated and compared to the curves of the virtually experimental tests under plane-stress conditions. Therefore, the values of total stress, back stress, drag stress, viscoplastic and plastic strain were collected at the maximum strain of the first cycle ($t = 5\,\mathrm{s}$). The results are presented in table 3.8 and in appendix B on B.1 and B.2, for cyclic strain ranges of ± 0.18, ± 0.11 and ± 0.14, respectively. However, only the results corresponding to a cyclic strain of ± 0.18 will be fully analyzed since it exceeds the cyclic strain used as training data.

With respect to the total stress for tensile tests (T_{xx}, T_{yy}) the correspondent stress (σ_{xx} and σ_{yy}) from the experimental tests is 324 MPa. Whereas the corresponding value by the proposed model has a difference in total stress of $\sigma_{xx} = 14.37\,\mathrm{MPa}$ in T_{xx} and $\sigma_{yy} = 7.59\,\mathrm{MPa}$ in T_{yy}. For simple shear test, the difference between predicted and experimental values of shear stress has a larger difference of 21.11 MPa. In addiction, from the experimental results, σ_{xx} and σ_{yy} should be 0 MPa, whereas the correspondent predicted values are 9.42 MPa and 28.42 MPa, respectively. Similarly, the back stress and viscoplastic strain have a smaller difference between predicted values and experimental results in tensile tests, when compared with simple shear testing. In cases where back stress and viscoplastic strain components should be 0, the model predicts a residual value from $-32.36\,\mathrm{MPa}$ to $36.03\,\mathrm{MPa}$ and -1.80×10^{-4} to -5.12×10^{-3}, respectively. In each integration the prediction of the derivatives is performed. Therefore, since these values result from several integrations until $t = 5\,\mathrm{s}$, if in each one present a prediction error, even if it is a small error, the integration will increase or decrease the variables ($\boldsymbol{\varepsilon}^{\mathrm{vp}}$, $\boldsymbol{\chi}$). Since the trained neural network does not present a $MSE = 0$ and $R^2 = 1$, when these values should be 0 they present a stack error.

In terms of plastic strain and drag stress, the predicted and experimental values are identical in tensile tests. However, in simple shear testing the plastic strain has a difference of 2.03×10^{-2} and drag stress a variation of 5.23 MPa.

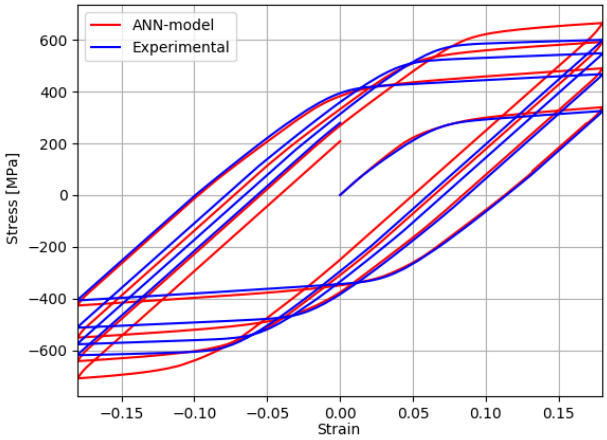
The complete reversed cyclic loadings, calculated using the machine learning model and the virtually experimental tests, are compared for tensile and simple shear tests. Figure 3.25 presents this comparison for a cyclic strain of ± 0.18. Both tensile tests shows that the predicted curve fits well with the experimental curve until the first two full cycles. The same is true for the simple shear testing.

Regarding the simulations performed with a cyclic strain of ± 0.11 and ± 0.14, the comparison of complete reversed cyclic loadings are presented in figure 3.26. By using both strains, it is possible to show that the trained model could predict well for the tensile and simple shear tests. Through these strains the correspondent results are better than using a cyclic strain of ± 0.18. This is due to the fact that the cyclic strain ranges used as training data were ± 0.05, ± 0.12, ± 0.16, thus the strains used to validate the model are expected to perform better within this range.

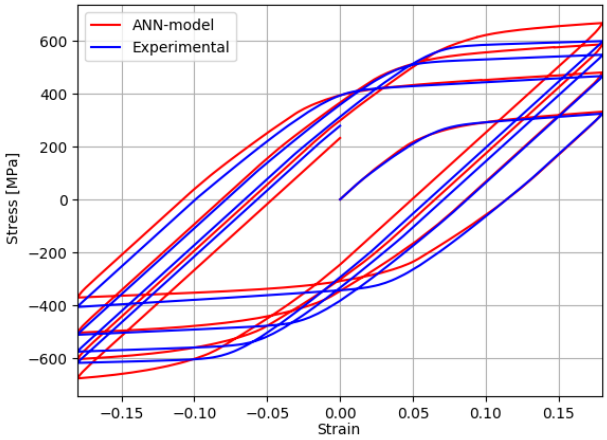
In conclusion, the proposed neural network model can reproduce the virtually experimental tests using Chaboche's viscoplastic material behavior. However, components of total stress, back stress and viscoplastic strain sometimes presents some residual values when these should be null.

Table 3.8: Values calculated with ANN-model in comparison with the virtually experimental values in 2D. These values correspond at the maximum strain of the first cycle ($t = 5\,\mathrm{s}$) on a strain range of ± 0.18.

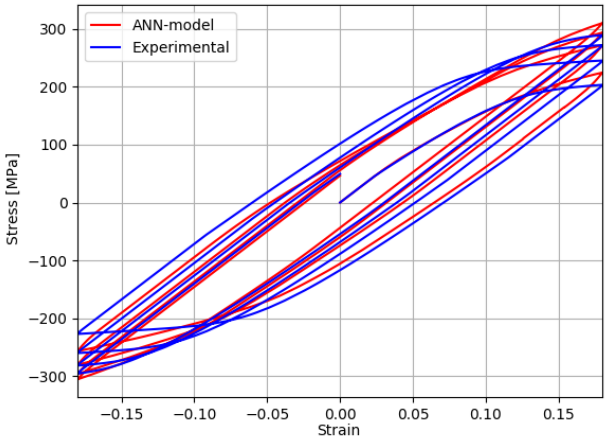
Strain range			± 0.18		
Testing			T_{xx}	T_{yy}	S_{xy}
Total Stress [MPa]	σ_{xx}	experimental	324.00	0	0
		ANN-model	338.37	0	9.42
	σ_{yy}	experimental	0	324.00	0
		ANN-model	0	331.59	28.42
	τ_{xy}	experimental	0	0	203.12
		ANN-model	7.07	3.12	224.23
Back Stress [MPa]	χ_{xx}	experimental	97.43	-97.43	0
		ANN-model	88.21	-83.84	-32.36
	χ_{yy}	experimental	-97.43	97.43	0
		ANN-model	-92.49	87.51	36.03
	χ_{xy}	experimental	0	0	137.46
		ANN-model	9.38	7.70	128.25
Viscoplastic Strain [-]	$\varepsilon_{xx}^{\mathrm{vp}}$	experimental	1.50×10^{-1}	-1.50×10^{-1}	0
		ANN-model	1.42×10^{-1}	-1.51×10^{-1}	-1.80×10^{-4}
	$\varepsilon_{yy}^{\mathrm{vp}}$	experimental	-1.50×10^{-1}	1.50×10^{-1}	0
		ANN-model	-1.33×10^{-1}	1.49×10^{-1}	-5.12×10^{-3}
	$\varepsilon_{xy}^{\mathrm{vp}}$	experimental	0	0	7.42×10^{-2}
		ANN-model	-3.68×10^{-3}	-1.62×10^{-3}	6.33×10^{-2}
Plastic Strain [-]	p	experimental	1.73×10^{-1}	1.73×10^{-1}	6.06×10^{-2}
		ANN-model	1.73×10^{-1}	1.75×10^{-1}	8.03×10^{-2}
Drag Stress [MPa]	R	experimental	94.32	94.32	66.07
		ANN-model	94.59	89.32	71.30



(a) Tensile test (T_{xx}) ±0.18

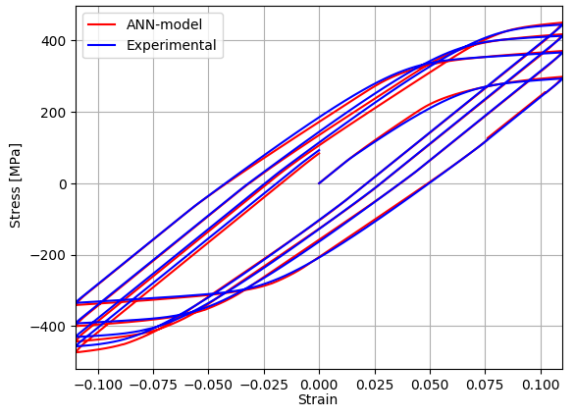


(b) Tensile test (T_{yy}) ±0.18

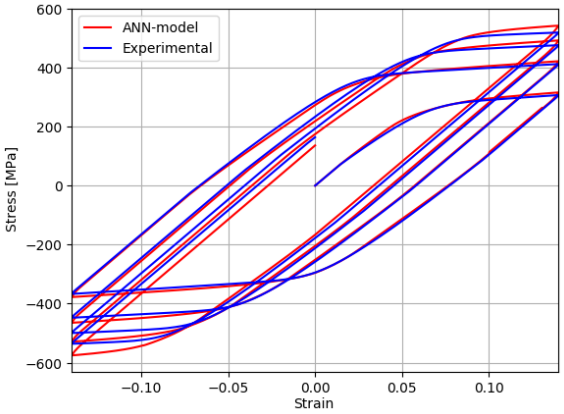


(c) Simple shear test (S_{xy}) ±0.18

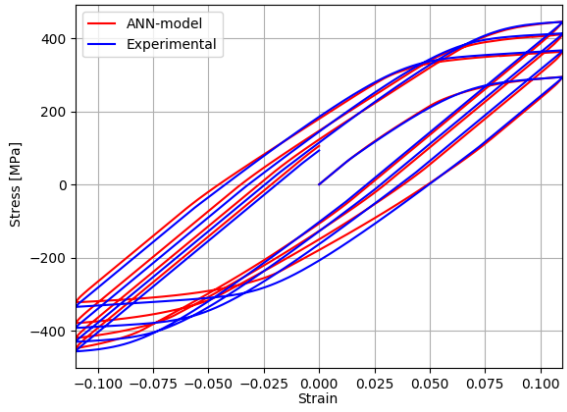
Figure 3.25: Comparison between predicted and experimental curves of 2D cyclic loading in tensile tests a) T_{xx}, b) T_{yy} and shear testing c) S_{xy}, with a cyclic strain of ±0.18.



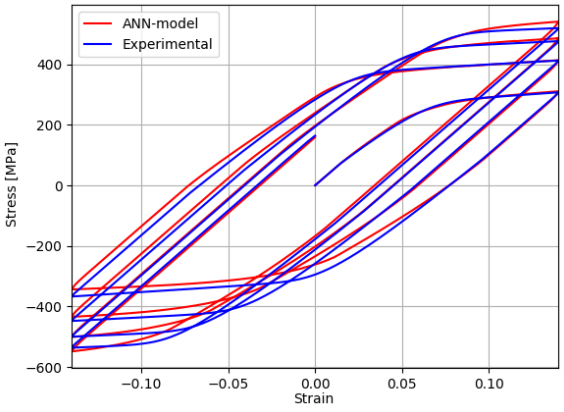
(a) Tensile test (T_{xx}) ±0.11



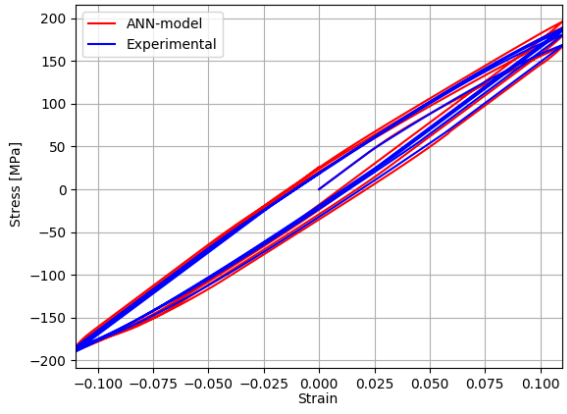
(b) Tensile test (T_{xx}) ±0.14



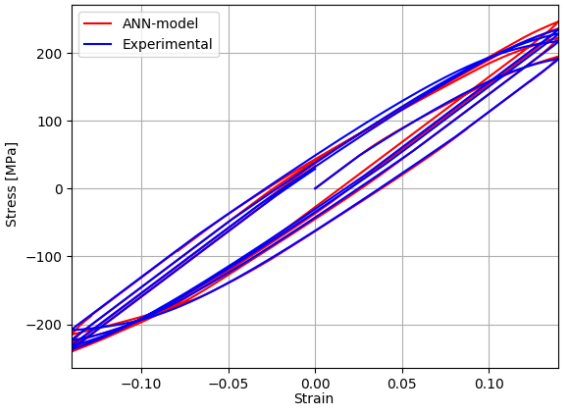
(c) Tensile test (T_{yy}) ±0.11



(d) Tensile test (T_{yy}) ±0.14



(e) Simple shear test (S_{xy}) ±0.11



(f) Simple shear test (S_{xy}) ±0.14

Figure 3.26: Comparison between predicted and experimental curves of 2D cyclic loading in tensile and simple shear testing. The left column corresponds to a cyclic strain of ±0.11 and the right to ±0.14.

Chapter 4

Implementation and analysis of the ANN-model on a FEA code

To verify that the trained model can mimic the constitutive material mechanical behavior in various geometries and boundary conditions, it's convenient to implement it on finite element analysis code. Abaqus [37] was the one selected because it has user routines [38] that allow the user to implement general constitutive models. One of its user routines is UMAT subroutine capability, where the mechanical behavior of a material can be described. In this case, the goal is to implement the trained neural network for 2D analysis on the UMAT subroutine.

ABAQUS/Standard analysis is divided into creating the model and submitting it for a job (simulation). In the first part, the following properties must be defined: part, material, assignment sections, assembly, steps, boundary conditions, loads and mesh. Note that, to use UMAT subroutine, the material definition must be specified as user defined. After defining all these properties, Abaqus creates a file (.inp extension) with all the necessary information about the desired model to be tested.

Submitting a job in Abaqus/Standard analysis begins with the definition of the initial conditions and the start of a step. Each analysis step is associated with a specific procedure that defines the type of analysis to be performed during the step, such as a static stress analysis. To obtain a converged solution inside the step, a loop is performed though iterations. In these subsequent iterations the corrections to the model's configuration are calculated using the stiffness from the end of the previous iteration until convergence. The flow of data and actions from the start of an Abaqus/Standard analysis to the end of a step is illustrated in figure 4.1.

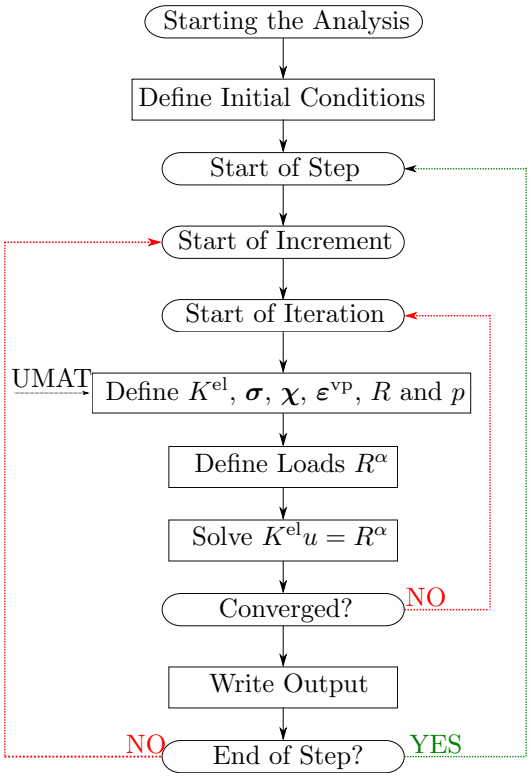


Figure 4.1: Flow of data and actions from the start of an ABAQUS/Standard analysis to the end of a step.

4.1 User defined subroutine UMAT and SDVINI

The UMAT subroutine capability can be used to define the mechanical constitutive behavior of a material. It is called at all material integration points of each element for which the material definition includes a user-defined material behavior. The UMAT can be used with any procedure that includes mechanical behavior and solution-dependent state variables. However, this subroutine has the following requirements:

- update the stresses and solution-dependent state variables to their values at the end of the increment for which it is called;
- provide the material Jacobian matrix for the mechanical constitutive model;
- be written in fortran, C or C++.

In this project, as mentioned before, instead of defining a constitutive behavior with classical equations, those are replaced with the trained machine learning model.

In order to initialize the solution-dependent state variables ($\boldsymbol{\chi}$, $\boldsymbol{\varepsilon}^{vp}$, R and p) with a specific value, the user subroutine SDVINI is used. This is used due to the fact that the material model used for training presents an initial drag stress of 50 MPa, as presented in equation 3.28 from subsection 3.1.2.1.

4.2 Communication between UMAT and machine learning model

A communication method must be implemented between the model trained in subsection 3.2.2.1 and the UMAT subroutine. The communication between the UMAT and a python 3.5 script is made through files.

Inside the UMAT subroutine (presented in C.3 from appendix C), the stress tensor and state variables are written into features.txt file, for each iteration. Then, a prediction has to be performed based on this features. For this purpose the estimator.pkl, scaler_x.pkl and scaler_y.pkl needs to be loaded. A call to `call_neuronalfem.py` is performed to clear the environment path, to therefore call `neuronalfem.py` program.

Inside the `neuronalfem.py` code (presented in C.4 from appendix C), the stress tensor and state variables are loaded from the UMAT and are fed to the neural network to output the correspondent derivatives. These derivatives are also written into a file for the UMAT to update the state variables and the stress tensor. The flow of data and actions are represented in figure 4.2.

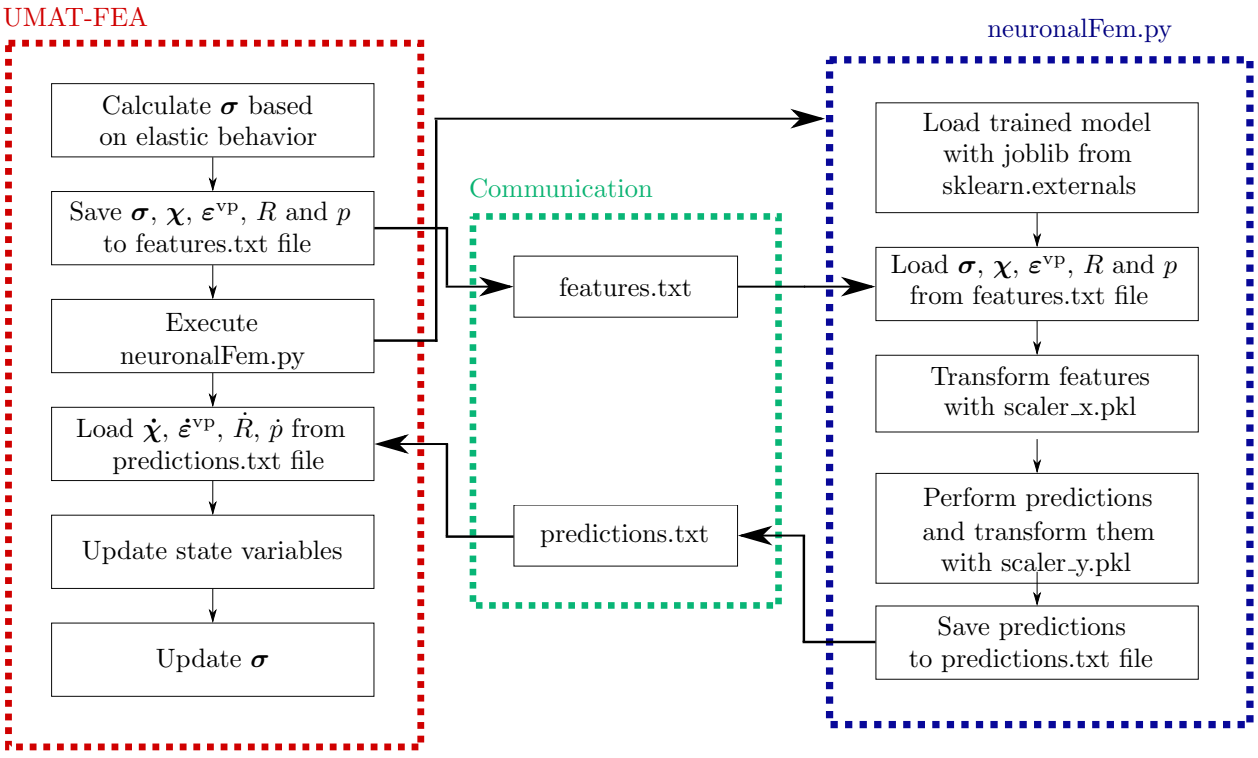
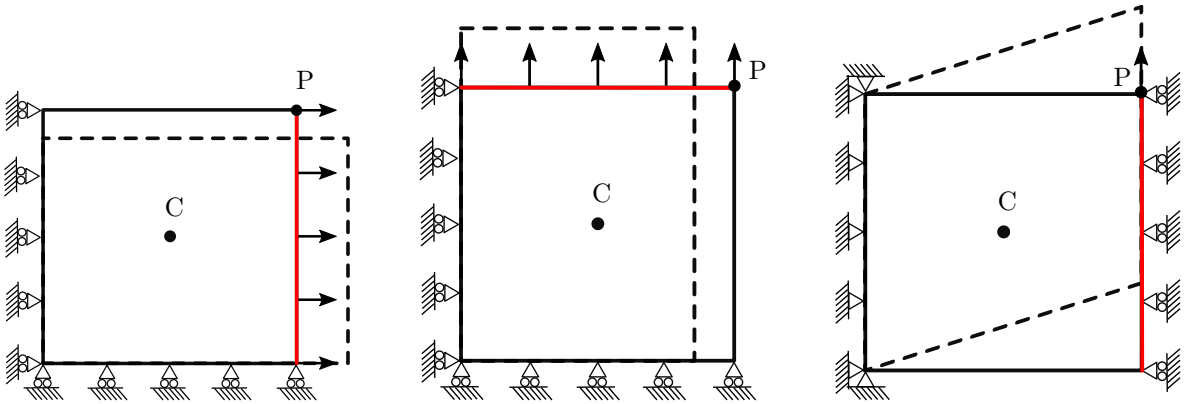


Figure 4.2: Communication method between the FEA and the machine learning model.

4.3 Validation

In order to validate the implementation previously described, it was simulated two tensile tests ($T_{xx} - FEA$, $T_{yy} - FEA$) and one simple shear ($S_{xy} - FEA$) for each strain on FEA code Abaqus.

To implement the tensile and simple shear tests on FEA, a geometry must be selected. For this purpose, a square with 1×1 mm is defined. Figure 4.3 represents each test with the corresponding boundary conditions. The red line in this figure represents the boundary condition for the mechanical displacement with a uniform distribution. In contrast to the validation made on section 3.2.2.2, the simulation performed on FEA was not set for cyclic displacement. For all tensile and simple shear tests the mechanical displacement used were 0.11 mm, 0.14 mm and 0.18 mm.



(a) Tensile test ($T_{xx} - FEA$) (b) Tensile test ($T_{yy} - FEA$) (c) Simple shear ($S_{xy} - FEA$)

Figure 4.3: Tests performed on FEA, as well as the boundary conditions for tensile tests in the a) xx and b) yy directions and c) for simple shear. Points C and P represent where the evaluation of stresses and strains are performed further.

Regarding the incrementation definition of the step, the time period was set to five seconds and the incrementation is fixed with an increment size of 0.05 s. The 4-node element CPS4, with plane-stress, was used. This definitions are described in table 4.1.

Table 4.1: Incrementation definitions for FEA step and element type to perform tensile and simple shear tests using the trained machine learning model (ANN-FEA) to predict the material behavior.

Time Period [s]	Incrementation	Increment Size [s]	Element Type
5	fixed	0.05	CPS4

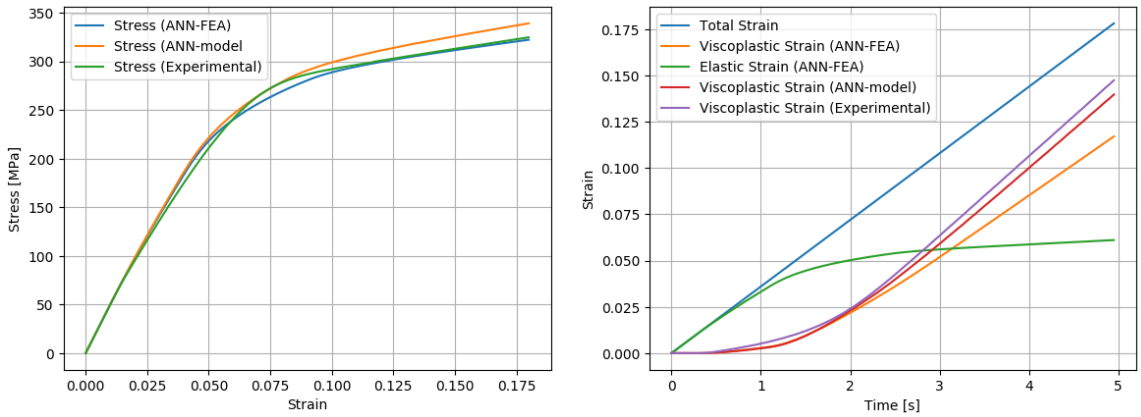
The comparison between the ANN-FEA, the ANN-model and the (virtually) experimental curves are only analyzed for a mechanical displacement of 0.18 mm, since it exceeds the total strain used as training data. Firstly, it is analyzed the normal stress and the strains for the tensile test T_{xx}, where the results are listed in table 4.2.

Table 4.2: Comparison between normal stress and strains obtained from ANN-FEA, ANN-model and experimental results for tensile test T_{xx}, as well as the percentage error for each neural network model.

	σ_{xx} [MPa]	ε^{t}_{xx} [-]	ε^{vp}_{xx} [-]	ε^{e}_{xx} [-]
ANN-FEA	322.13	1.80×10^{-1}	1.17×10^{-1}	6.30×10^{-2}
ANN-model	338.37	1.80×10^{-1}	1.42×10^{-1}	3.80×10^{-2}
Experimental	324.00	1.80×10^{-1}	1.50×10^{-1}	3.00×10^{-2}
Error [%]	0.58/4.44	-	22.00/5.33	110.00/26.67

For this test, the predicted stress-strain curve determined using ANN-FEA fits the experimental curve more accurately than the predicted curve using ANN-model. For example, at the maximum strain ($t = 5\,\text{s}$), the stress from experimental results has a value of 324 MPa, whereas the predicted values from ANN-FEA and ANN-model are 322.13 MPa and 338.37 MPa, respectively. Figure 4.4(a) shows the comparison between both curves for the centroid (C point) of the element.

The evolution of the strain during the analysis of the P point (fourth point of integration) are illustrated on figure 4.4(b). In this figure, the predicted viscoplastic strain from ANN-FEA until $t = 2\,\text{s}$ behaves like the ANN-model, however, at the end of the test they present a percentage error of 22.00 % and 5.33 %, respectively, when comparing to the experimental values.



(a) $\sigma_{xx} - \varepsilon^{t}_{xx}$ curves from T_{xx} test .

(b) $\varepsilon^{vp}_{xx}, \varepsilon^{t}_{xx}, \varepsilon^{e}_{xx}$ over time from T_{xx} test.

Figure 4.4: Stress and strain curves obtained from proposed model of the ANN-FEA and ANN-model and from experimental results for tensile test (T_{xx}).

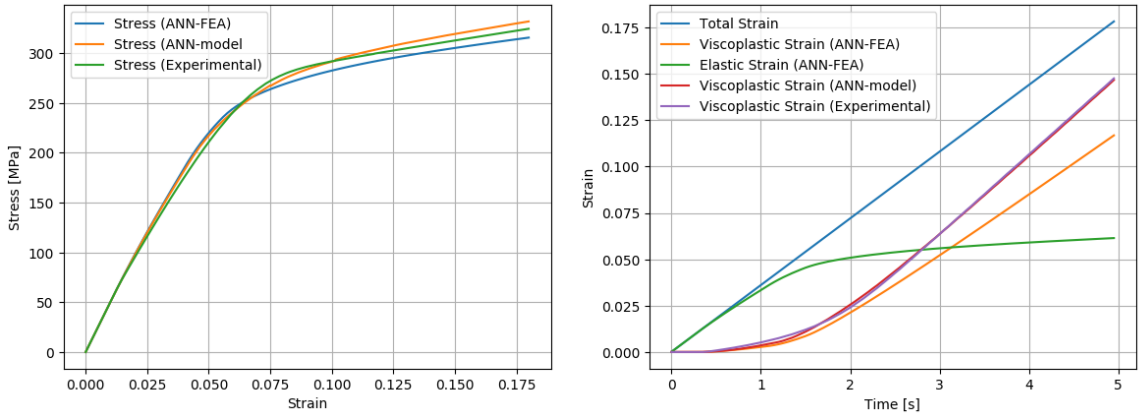
For the T_{yy} test , the stress and strain values at maximum strain are presented in table 4.3.

In contrast to the test T_{xx}'s results, both predicted curves present nearly the same normal stress gap at maximum strain. The percentage error between the ANN-FEA and ANN-model curves with the experimental curve is 2.55 % and 2.34 %, respectively. Figure 4.5(a) shows the comparison of these curves.

Table 4.3: Comparison between normal stress and strains obtained from ANN-FEA, ANN-model and experimental results for tensile test T_{yy}, as well as the percentage error for each neural network model.

	σ_{yy} [MPa]	ε^{t}_{yy} [-]	ε^{vp}_{yy} [-]	ε^{e}_{yy} [-]
ANN-FEA	315.72	1.80×10^{-1}	1.17×10^{-1}	6.30×10^{-2}
ANN-model	331.59	1.80×10^{-1}	1.49×10^{-1}	3.10×10^{-2}
Experimental	324.00	1.80×10^{-1}	1.50×10^{-1}	3.00×10^{-2}
Error [%]	2.55/2.34	-	22.00/0.67	110.00/3.33

The strains evolution during the analysis for the P point are illustrated in figure 4.5(b). From this figure, it can be observed that the predicted viscoplastic strain from ANN-model perfectly matches the experimental curve as of $t = 2.5\,\mathrm{s}$ to $t = 5\,\mathrm{s}$. Although, the predicted viscoplastic strain from ANN-FEA is slightly lower when compared to the other two, presenting a percentage error of 22 %.



(a) $\sigma_{yy} - \varepsilon^{t}_{yy}$ curves from T_{yy} test.

(b) $\varepsilon^{vp}_{yy}, \varepsilon^{t}_{yy}, \varepsilon^{e}_{yy}$ over time from T_{yy} test.

Figure 4.5: Stress and strain curves obtained from proposed model of the ANN-FEA and ANN-model and from experimental results for tensile test (T_{yy}).

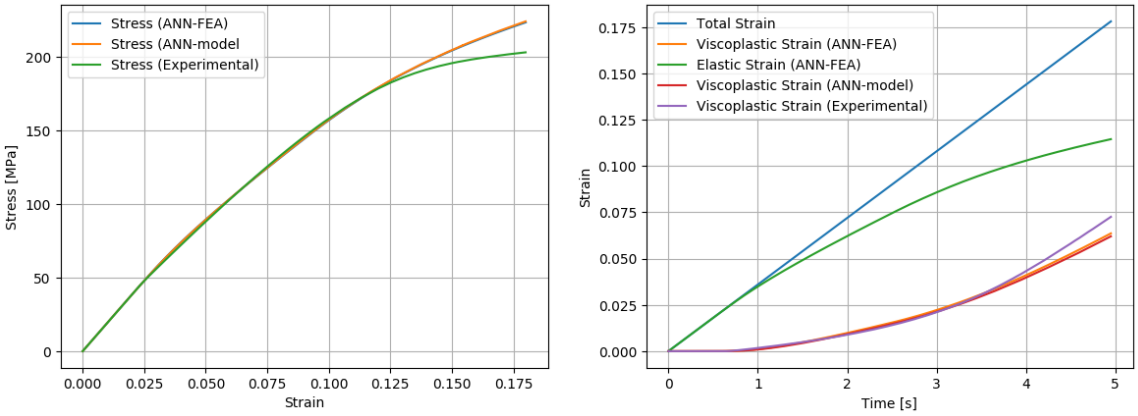
For the simple shear test (S_{xy}), the stress and strain values at maximum strain are presented on table 4.4.

The shear stress are properly predicted until the mechanical displacement hits 0.125, when comparing the values predicted from the ANN-FEA and ANN-model with the experimental results. At maximum strain, they reach 224.13 MPa, 223.76 MPa and 203.12 MPa, respectively.

The evolution of the strains during the analysis are illustrated on figure 4.6(b). From this figure it can be observed that both predicted viscoplastic strains perfectly matches the experimental curve until $t = 3\,\mathrm{s}$, but from then on, they drive way from it. At the maximum mechanical displacement, viscoplastic strain from ANN-FEA, ANN-model and experimental results reach 6.33×10^{-2}, 6.36×10^{-2} and 7.42×10^{-2}, respectively.

Table 4.4: Comparison between normal stress and strains obtained from ANN-FEA, ANN-model and experimental results for tensile test S_{xy}, as well as the percentage error for each neural network model.

	τ_{xy} [MPa]	ε^{t}_{xy} [-]	ε^{vp}_{xy} [-]	ε^{e}_{xy} [-]
ANN-FEA	223.76	1.80×10^{-1}	6.36×10^{-2}	1.16×10^{-2}
ANN-model	224.23	1.80×10^{-1}	6.33×10^{-2}	1.17×10^{-2}
Experimental	203.12	1.80×10^{-1}	7.42×10^{-2}	1.06×10^{-2}
Error [%]	10.16/10.39	-	14.29/14.69	9.43/10.38



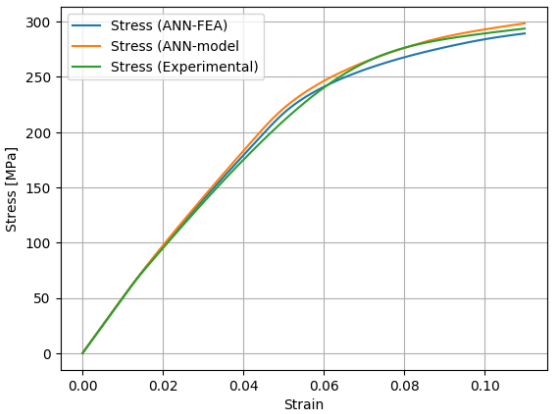
(a) $\tau_{xy} - \varepsilon^{t}_{xy}$ curves from S_{xy} test.

(b) $\varepsilon^{vp}_{xy}, \varepsilon^{t}_{xy}, \varepsilon^{e}_{xy}$ over time from S_{xy} test.

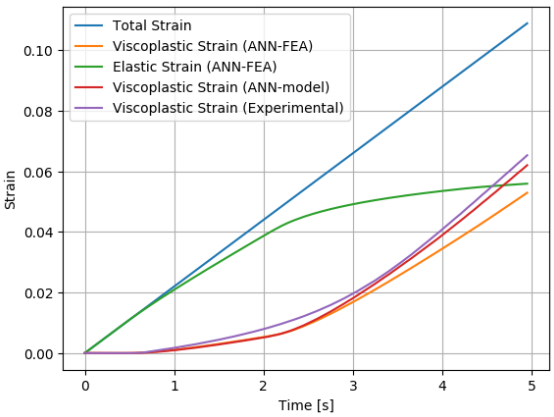
Figure 4.6: Stress and strain curves obtained from proposed model of the ANN-FEA and ANN-model and from experimental results for simple shear test (S_{xy}).

On the tensile tests, the same neural network model seems to predict differently whether it's used as ANN-model or ANN-FEA. However, this occur due to the fact that the integration method used is different. The ANN-model uses a Runge-Kutta scheme whereas the ANN-FEA uses a total Lagrangian implicit scheme to the time integration. In addiction, Runge-Kutta uses error control to integrate, whereas in FEA, for this experiment, the increment was set to fixed with no error control. Therefore, the results obtained with the same neural network model may vary according to the size of increments and whether or not error control is used.

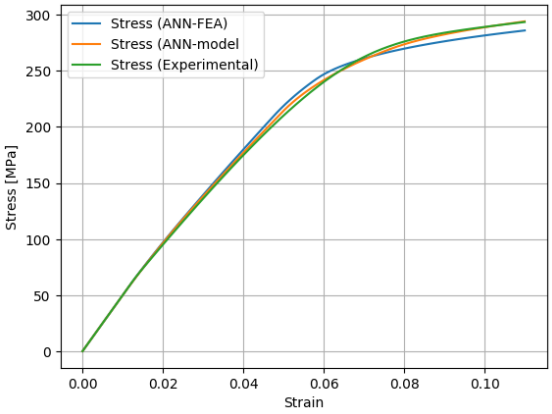
The results of the previous strain are similar for a mechanical displacement of 0.11 and 0.14. The results are presented on figures 4.7 and 4.8, respectively. On the left of each figure is presented the comparison of the normal stress for the tests T_{xx}, T_{yy} and S_{xy}. On the right the evolution of total, elastic and plastic strain can be visualized.



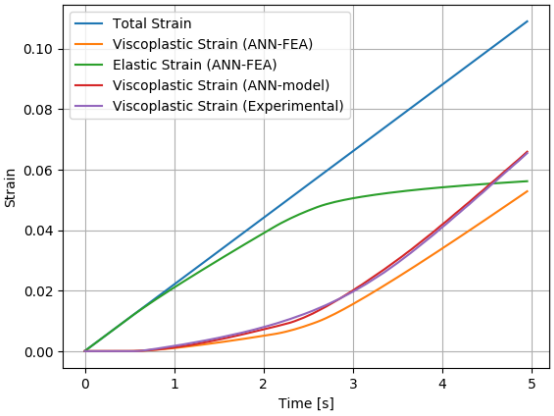
(a) $\sigma_{xx} - \varepsilon^{\mathrm{t}}_{xx}$ curves from T_{xx} test.



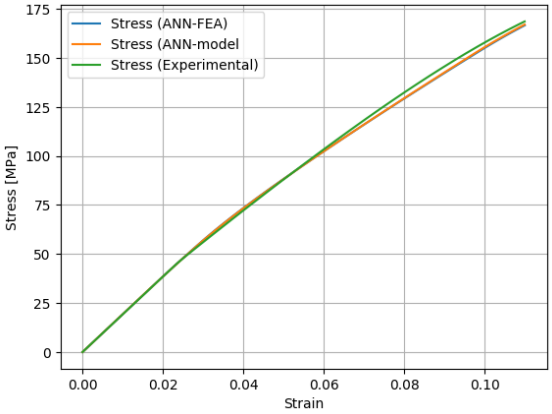
(b) $\varepsilon^{\mathrm{vp}}_{xx}, \varepsilon^{\mathrm{t}}_{xx}, \varepsilon^{\mathrm{e}}_{xx}$ over time from T_{xx} test.



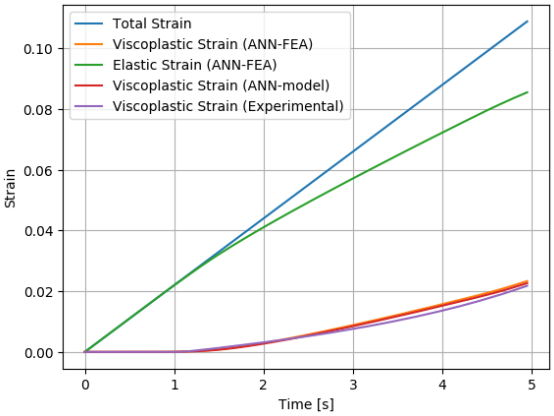
(c) $\sigma_{yy} - \varepsilon^{\mathrm{t}}_{yy}$ curves from T_{yy} test.



(d) $\varepsilon^{\mathrm{vp}}_{yy}, \varepsilon^{\mathrm{t}}_{yy}, \varepsilon^{\mathrm{e}}_{yy}$ over time from T_{yy} test.

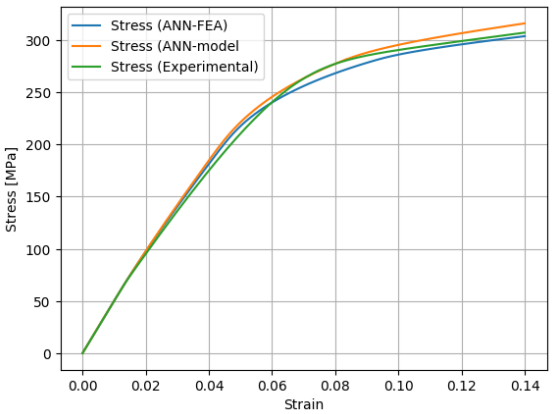


(e) $\tau_{xy} - \varepsilon^{\mathrm{t}}_{xy}$ curves from S_{xy} test.

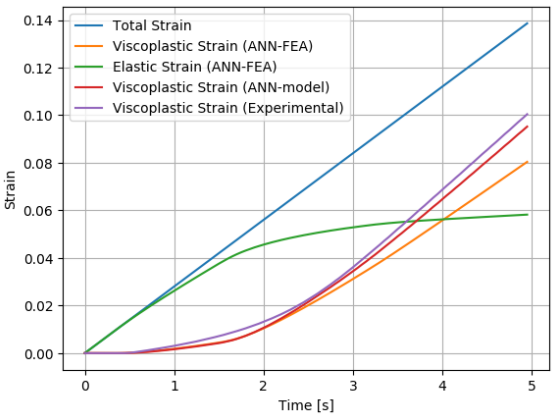


(f) $\varepsilon^{\mathrm{vp}}_{xy}, \varepsilon^{\mathrm{t}}_{xy}, \varepsilon^{\mathrm{e}}_{xy}$ over time from S_{xy} test.

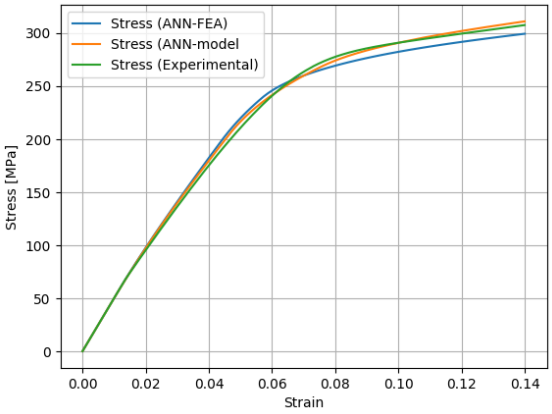
Figure 4.7: Stress and strain curves obtained with ANN-FEA, ANN-model and experimental results for tests T_{xx}, T_{yy} and S_{xy} with a maximum strain of 0.11.



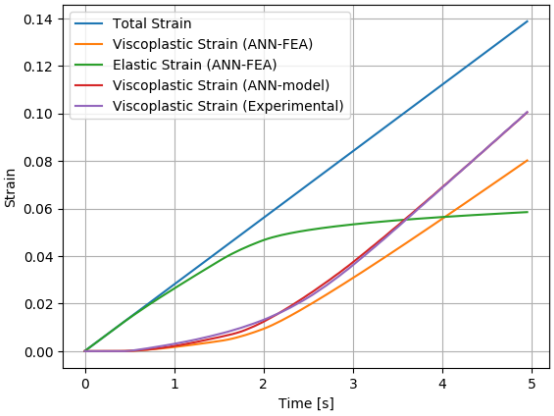
(a) $\sigma_{xx} - \varepsilon^{\mathrm{t}}_{xx}$ curves from T_{xx} test.



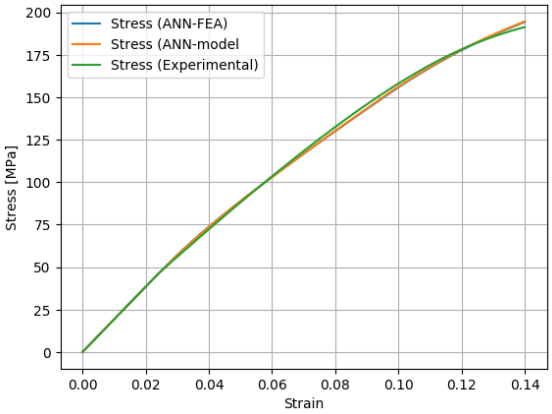
(b) $\varepsilon^{\mathrm{vp}}_{xx}, \varepsilon^{\mathrm{t}}_{xx}, \varepsilon^{\mathrm{e}}_{xx}$ over time from T_{xx} test.



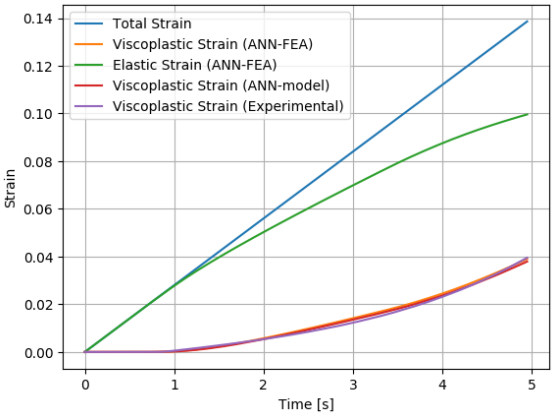
(c) $\sigma_{yy} - \varepsilon^{\mathrm{t}}_{yy}$ curves from T_{yy} test.



(d) $\varepsilon^{\mathrm{vp}}_{yy}, \varepsilon^{\mathrm{t}}_{yy}, \varepsilon^{\mathrm{e}}_{yy}$ over time from T_{yy} test.



(e) $\tau_{xy} - \varepsilon^{\mathrm{t}}_{xy}$ curves from S_{xy} test.



(f) $\varepsilon^{\mathrm{vp}}_{xy}, \varepsilon^{\mathrm{t}}_{xy}, \varepsilon^{\mathrm{e}}_{xy}$ over time from S_{xy} test.

Figure 4.8: Stress and strain curves obtained with ANN-FEA, ANN-model and experimental results for tests T_{xx}, T_{yy} and S_{xy} with a maximum strain of 0.14.

Chapter 5

Testing machine learning model on complex geometry

5.1 Adapted butterfly test

On the Last Chapter it is demonstrated that the trained neural network model is capable of reproducing the virtually experimental results for 2D plane-stress conditions on FEA. The final step is to test the proposed model on a more complex geometry. For this purpose, the adapted butterfly specimen is selected [39]. On a first step, the adapted butterfly geometry must be defined. Figure 5.1 illustrates the specimen's geometry in millimeters.

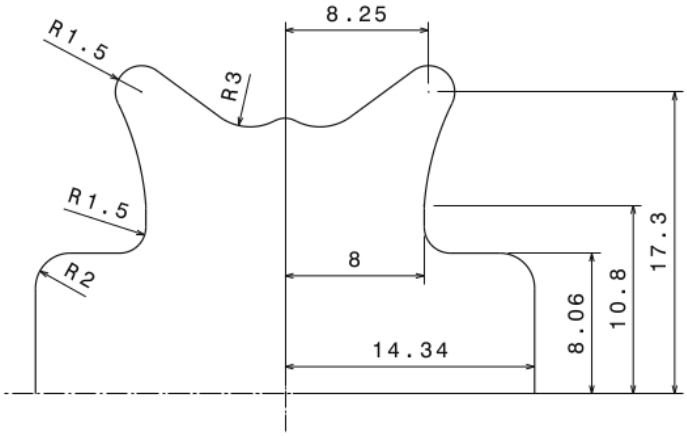
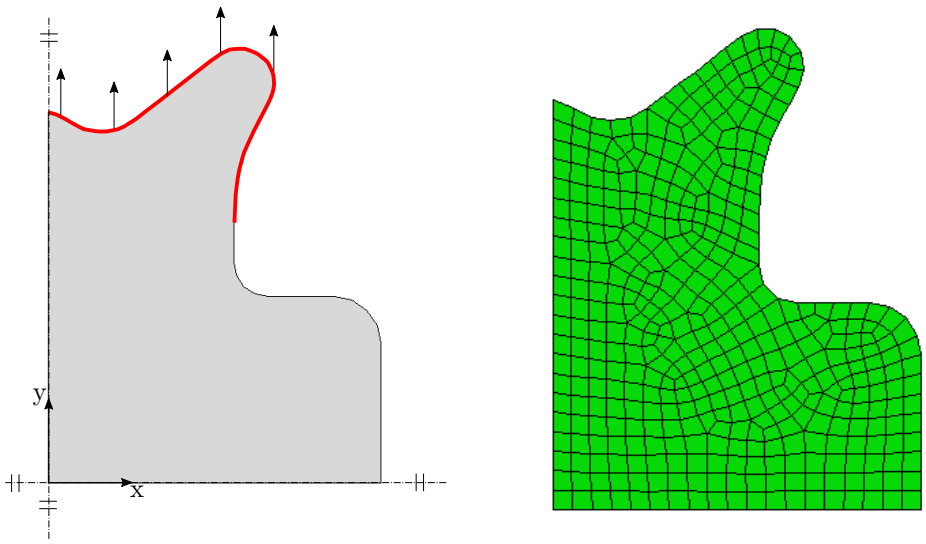


Figure 5.1: Detail geometry of adapted butterfly specimen.

The purpose of this test is to perform a tensile test T_{yy}, in order to simulate a tool displacement. Thus, a mechanical displacement is imposed to the specimen's top. The boundary conditions are illustrated on figure 5.2(a), where the symmetry along X and Y axis are represented, as well as the mechanical displacement (line in red), with a tool displacement of 9.6 mm.

Due to the symmetry boundary conditions, it is used only one fourth of the specimen, which is meshed with 2D 4-node bilinear elements (CPS4). In order to reduce the simulation time, it is used an unstructured mesh with an element size of 0.75 mm as shown in figure 5.2(b).

Regarding the incrementation definition of the step, the time period was set to five seconds in resemblance to the tests made on last section, whereas the incrementation is fixed but with an increment size of 0.2 s.



(a) Tensile test (xx). (b) Numerical mesh.

Figure 5.2: Representation of a) boundary conditions and b) numerical mesh for the adapted butterfly specimen.

5.2 Results

With all the parameters defined, the job is submitted using the UMAT subroutine in appendix C.3 to define the mechanical constitutive behavior. The wallclock time in seconds to complete the analysis was 92 502 s, which correspond approximately to twenty five hours and forty two minutes.

To represent the obtained results, the contour presented in figure 5.3 represents the von Mises equivalent stress. In this image it is also demonstrated the three elements chosen to evaluate the performance of the proposed model. In addiction, the evaluation is performed only for the second integration point for all chosen elements.

To evaluate the performance of the proposed model in this test, it is necessary to generate virtually experimental results using the Chaboche constitutive model. Thus, the total strain field and step time from all selected points are collected and used as total strain and time period for algorithm 2, in order to generate virtually experimental results. This provides the values necessary to compare the results obtained in this test, using the ANN, with the experimental results. It is important to mention that the ANN model was trained for three specific test cases. Which in the case of tensile tests, shear stress was always zero, as well as the shear strain.

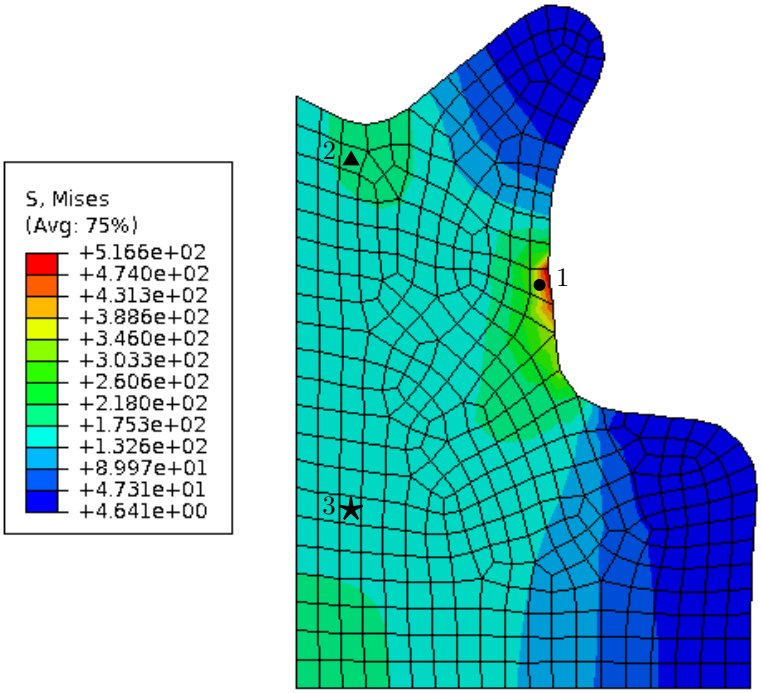


Figure 5.3: Contour obtained for the von Mises equivalent stress and the elements chosen for evaluation.

The values of total, back and drag stress are listed in table 5.1, at the end of the step, which is responsible for the mechanical displacement. Here it can be verified that, in general, the proposed model show stress values close to the experimental values. However, for the first point, σ_{xx} and τ_{xy} presents a difference of 40.46 MPa and 31.25 MPa, respectively. Although, for the same point, σ_y only presents a difference of 4.04 MPa. This divergence of results from total stress can be explained by the fact that the combination of total strain's components being $(-0.0914,\ 0.2008,\ 0.0442)$. The artificial neural network was never trained with the existence of displacement in all three components of total strain. In addiction, the machine learning model was only trained for a maximum strain of 0.16. From table 5.1, χ_{xy} stands negatively for the first point, as the predicted value from ANN is 3.49 MPa, whereas the expected value is 23.94 MPa.

For the second and third points, due to the lower mechanical displacement imposed , when compared to the first point, the performance of ANN is good , with anything to emphasize.

With regard to the viscoplastic and plastic strains, the values at the end of the step for all selected points are presented in table 5.2. Overall, the performance of ANN is good, and only two values are worth mention negatively. For the first and third points, ANN predicted $\varepsilon_{xy}^{\text{vp}}$ as 8.266×10^{-3} and 9.220×10^{-5}, whereas the expected values are 2.452×10^{-3} and 5.400×10^{-5}, respectively.

Table 5.1: Comparison between ANN and experimental values of $\boldsymbol{\sigma}$, $\boldsymbol{\chi}$ and R, as well as the percentage error between them, at the end of the step for the three selected points.

		σ_{xx} [MPa]	σ_{yy} [MPa]	τ_{xy} [MPa]	χ_{xx} [MPa]	χ_{yy} [MPa]	χ_{xy} [MPa]	R [MPa]
Point 1 ●	*ANN*	283.88	533.31	69.08	-100.18	101.41	3.49	81.97
	Experimental	243.42	537.35	37.83	-95.94	95.94	23.94	82.81
	Error[%]	16.62	0.75	82.61	4.42	5.7	85.42	1.01
Point 2 ▲	*ANN*	-29.96	180.21	0.96	-64.34	64.08	1.72	52.49
	Experimental	-25.78	180.33	1.05	-69.35	69.35	0.77	53.35
	Error[%]	0.16	0.07	8.57	7.22	7.60	0.01	1.61
Point 3 ★	*ANN*	0.62	163.06	7.47	-48.62	48.28	1.25	51.80
	Experimental	-0.66	168.67	6.61	-52.11	52.11	4.18	52.07
	Error[%]	193.94	3.33	13.01	6.7	7.35	70.1	0.52

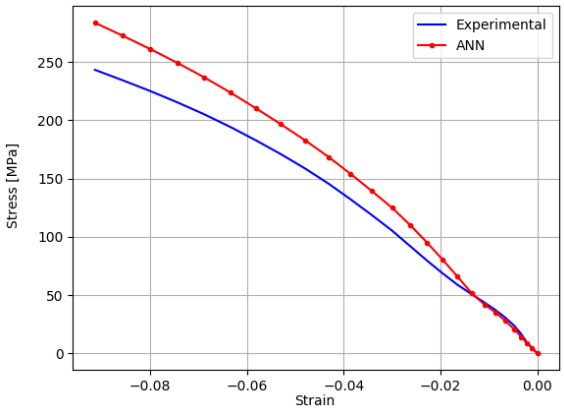
Table 5.2: Comparison between ANN and experimental values of $\boldsymbol{\varepsilon}^{\mathrm{vp}}$ and p, as well as the percentage error between them, at the end of the step for the three selected points.

		$\varepsilon_{xx}^{\mathrm{vp}}$	$\varepsilon_{yy}^{\mathrm{vp}}$	$\varepsilon_{xy}^{\mathrm{vp}}$	p
Point 1 ●	*ANN*	-1.162×10^{-1}	1.111×10^{-1}	8.266×10^{-3}	1.313×10^{-1}
	Experimental	-1.079×10^{-1}	1.079×10^{-1}	2.452×10^{-2}	1.262×10^{-1}
	Error[%]	7.69	2.97	66.29	4.04
Point 2 ▲	*ANN*	-9.948×10^{-3}	1.055×10^{-2}	1.798×10^{-4}	1.162×10^{-2}
	Experimental	-1.077×10^{-2}	1.077×10^{-2}	1.400×10^{-4}	1.244×10^{-2}
	Error[%]	7.63	2.04	28.43	6.59
Point 3 ★	*ANN*	-7.230×10^{-3}	7.836×10^{-3}	9.220×10^{-5}	8.538×10^{-3}
	Experimental	-6.630×10^{-3}	6.630×10^{-3}	5.400×10^{-4}	7.670×10^{-3}
	Error[%]	9.05	18.19	82.93	11.32

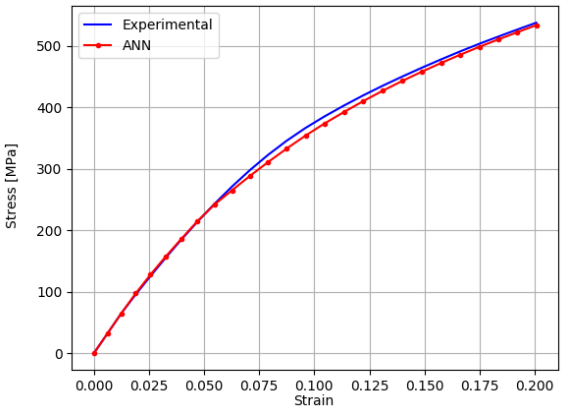
It's important to evaluate the evolution of $\boldsymbol{\sigma}$ between the ANN with the experimental results. Figure 5.4 illustrates this evaluation for the first selected point. It's observed that for both σ_{xx} and σ_{yy} from ANN curves fit well the experimental results, whereas the τ_{xy} from ANN curve fails to predict the experimental curve. This is due to the fact that the relationship between stress-strain was never used for training. State variables: Back stress, viscoplastic strain, drag stress and plastic strain for the first selected point were also evaluated in terms of evolution over time and these are represented by figures 5.5, 5.6, 5.7 and 5.8, respectively. It's also observed that for all state variables the ANN curves fit well the experimental ones.

For the second and third selected points, both stress and state variables evolutions are illustrated on figures 5.9 and 5.10, 5.11 and 5.12, respectively. From these figures, the maximum strain reached for all points are below 0.05 and the ANN curves fit well the experimental ones.

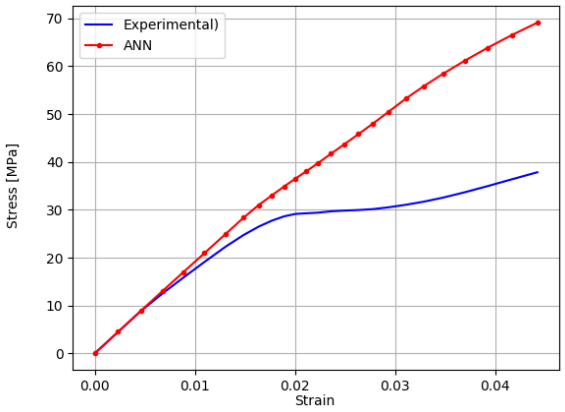
In conclusion, the proposed machine learning model, even with different combinations of strain never used in training, is capable of reproducing the material behavior, defined by Chaboche's constitutive model. In addiction, ANN have no problems within a simulation with a more complex geometry and several elements.



(a) $\sigma_{xx} - \varepsilon^{t}_{xx}$ curves.



(b) $\sigma_{yy} - \varepsilon^{t}_{yy}$ curves.



(c) $\tau_{xy} - \varepsilon^{t}_{xy}$ curves.

Figure 5.4: Stress tensor $\boldsymbol{\sigma}$ in function of it's correspondent strain, obtained from ANN and experimental values, for the first selected point.

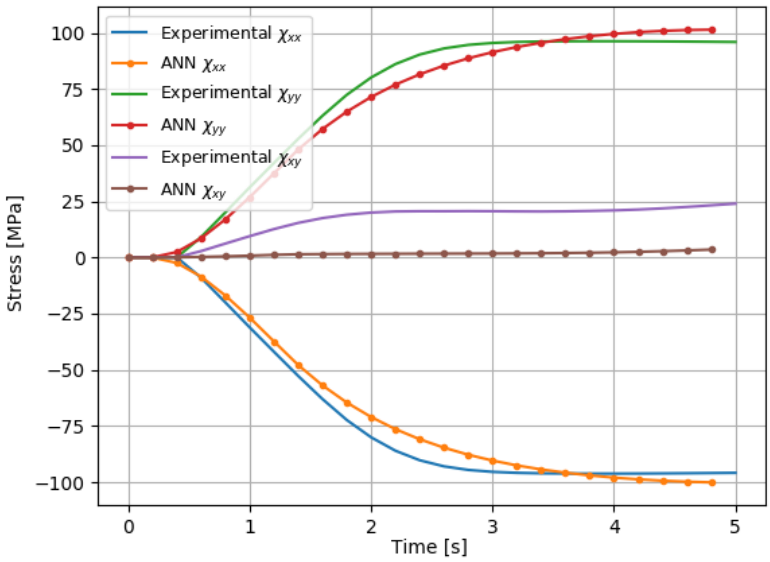


Figure 5.5: Back stress tensor $\boldsymbol{\chi}$ over time, obtained from ANN and the experimental values, for the first selected point.

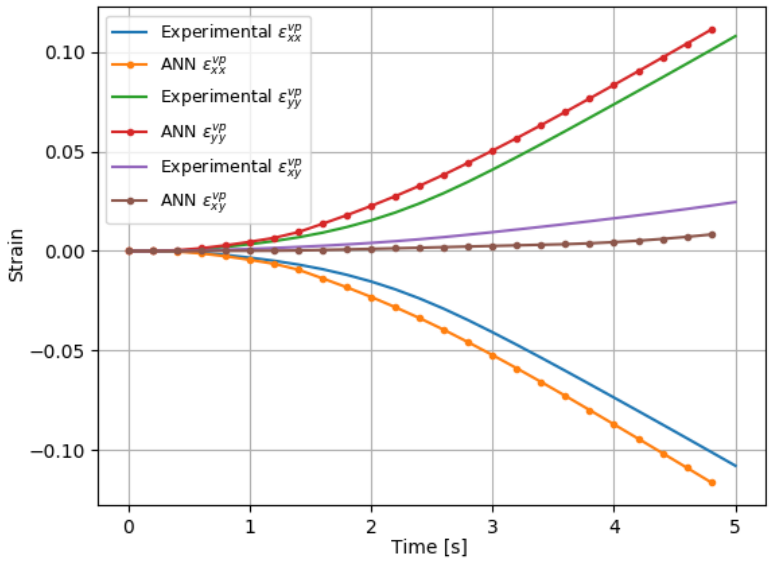


Figure 5.6: Viscoplastic strain tensor $\boldsymbol{\varepsilon}^{\mathrm{vp}}$ over time, obtained from ANN and the experimental values, for the first selected point.

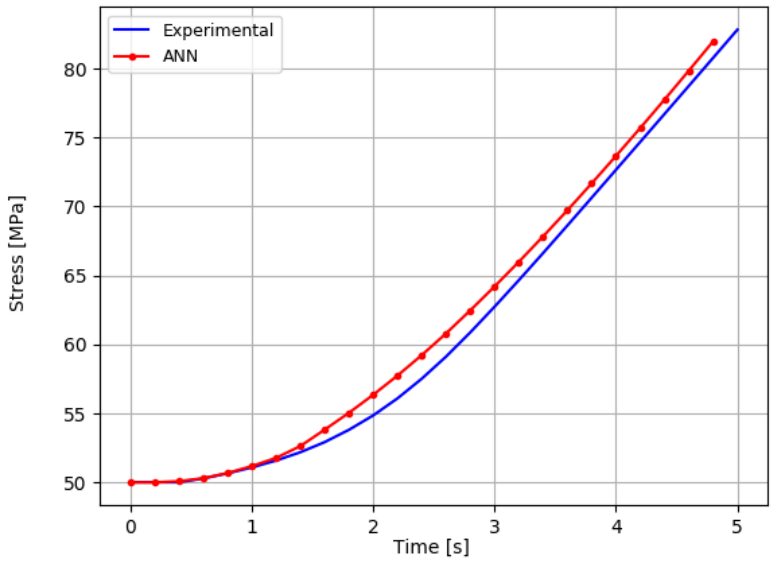


Figure 5.7: Drag stress R over time, obtained from ANN and the experimental values, for the first selected point.

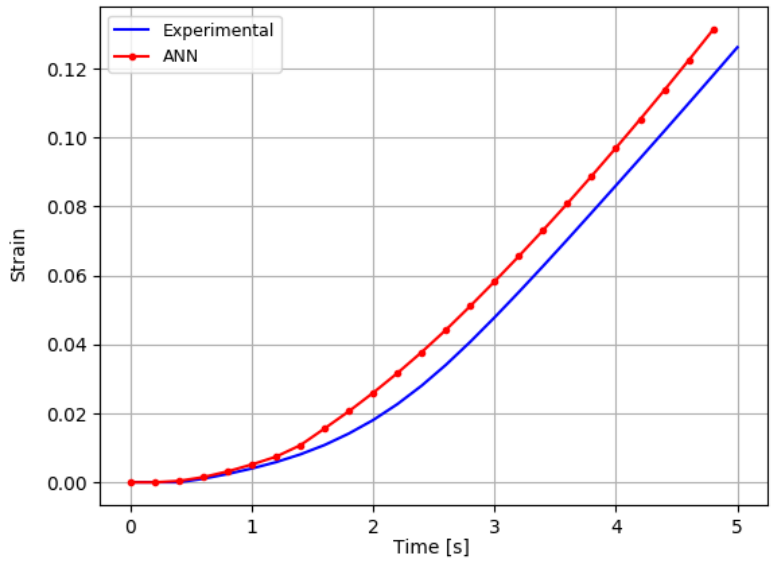
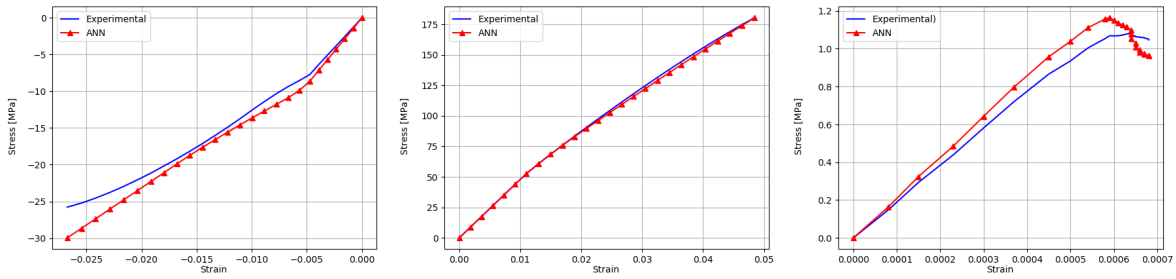
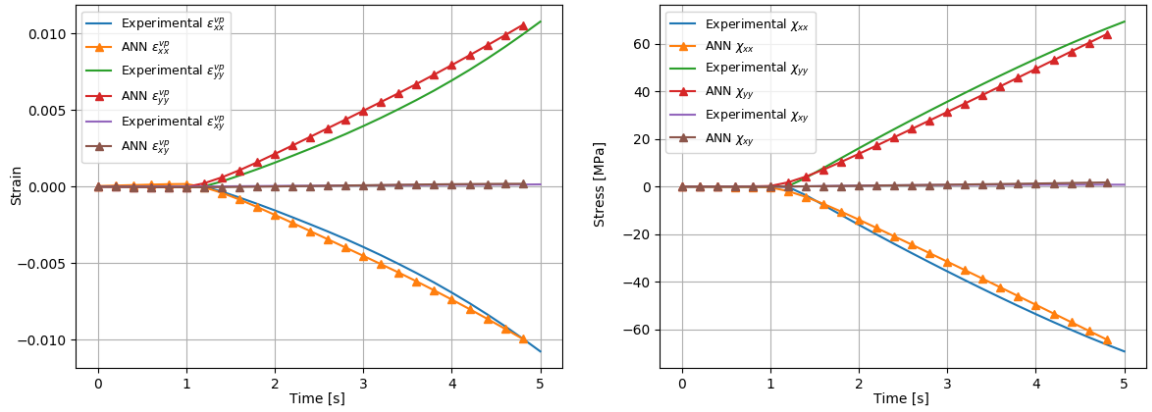


Figure 5.8: Plastic strain p over time, obtained from ANN and the experimental values, for the first selected point.



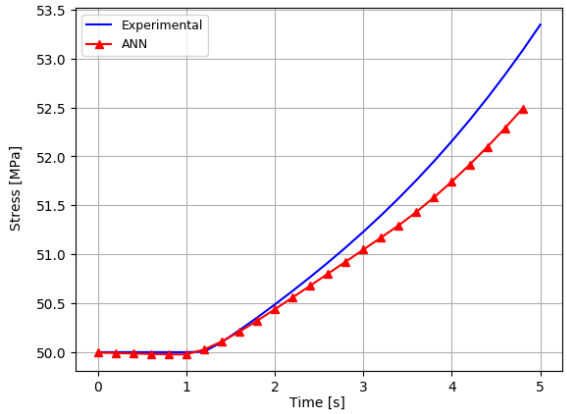
(a) $\sigma_{xx} - \varepsilon^{\mathrm{t}}_{xx}$ curves. (b) $\sigma_{yy} - \varepsilon^{\mathrm{t}}_{yy}$ curves. (c) $\tau_{xy} - \varepsilon^{\mathrm{t}}_{xy}$ curves.

Figure 5.9: Stress tensor $\boldsymbol{\sigma}$ in function of it's correspondent strain, obtained from ANN and experimental values, for the second selected point.

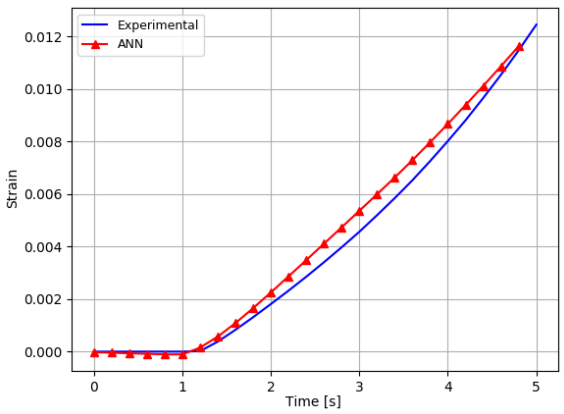


(a) Viscoplastic Strain tensor $\boldsymbol{\varepsilon}^{\mathrm{VP}}$.

(b) Back Stress tensor $\boldsymbol{\chi}$.

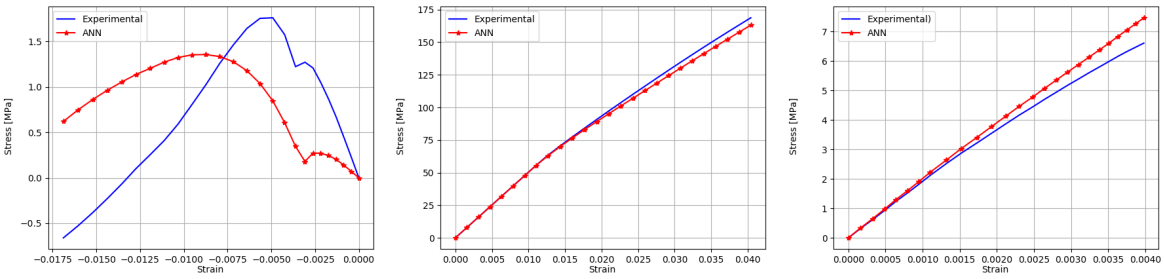


(c) Drag Stress R.



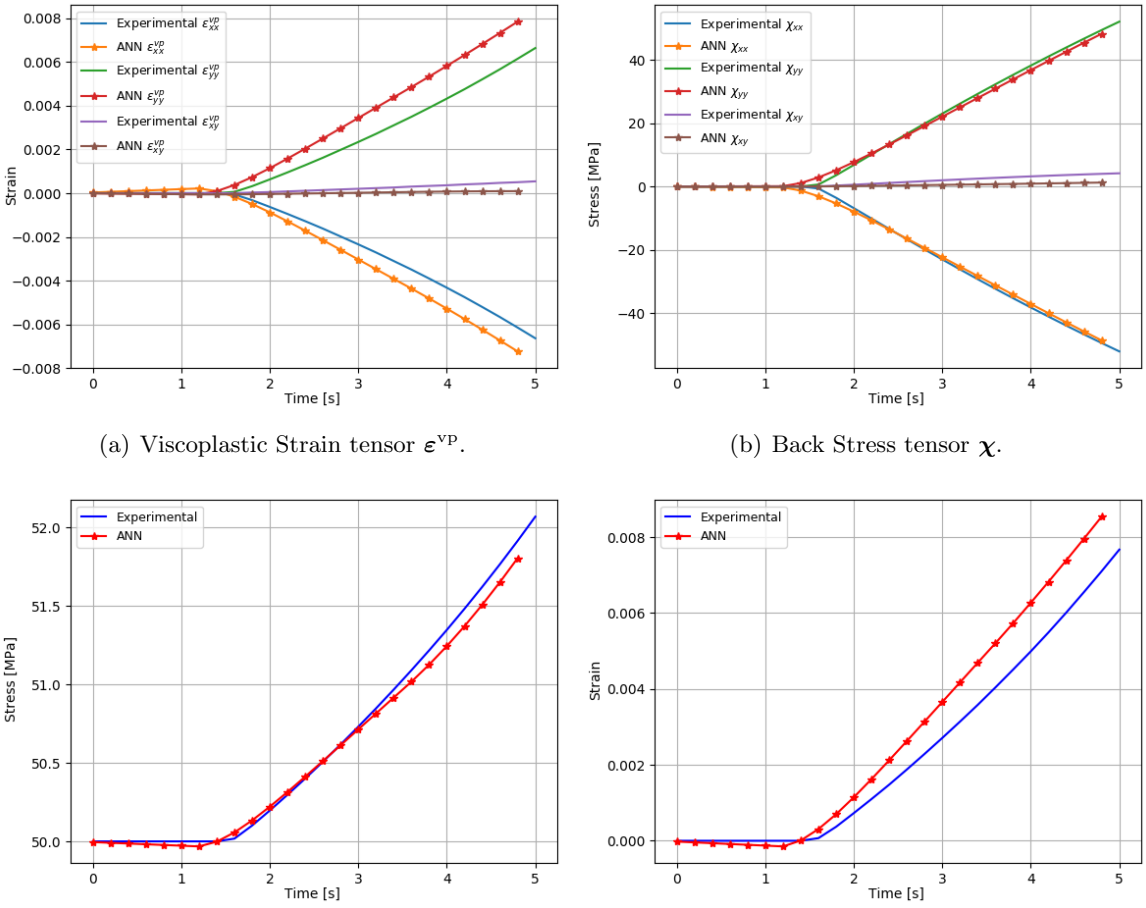
(d) Plastic Strain p.

Figure 5.10: State variables over time, obtained from ANN and the experimental values, for the second selected point.



(a) $\sigma_{xx} - \varepsilon^{\mathrm{t}}_{xx}$ curves. (b) $\sigma_{yy} - \varepsilon^{\mathrm{t}}_{yy}$ curves. (c) $\tau_{xy} - \varepsilon^{\mathrm{t}}_{xy}$ curves.

Figure 5.11: Stress tensor $\boldsymbol{\sigma}$ in function of it's correspondent strain, obtained from ANN and experimental values, for the third selected point.



(a) Viscoplastic Strain tensor $\boldsymbol{\varepsilon}^{\mathrm{VP}}$. (b) Back Stress tensor $\boldsymbol{\chi}$.

(c) Drag Stress R. (d) Plastic Strain p.

Figure 5.12: State variables over time, obtained from ANN and the experimental values, for the third selected point.

Chapter 6

Conclusions and Future Work

During the course of this work, a constitutive model using artificial intelligence techniques was accomplished, in order to reproduce an explicit constitutive model. This was successfully reproduced for 1D and 2D (plane-stress) conditions, thus achieving the main objective. With regard to the training process for both conditions, it was demonstrated that for only one hidden layer, the number of hidden neurons increase from four to sixteen. In spite of this grow being non-linear, it shows that in order to predict the rates of viscoplastic strain, plastic strain, kinematic and isotropic hardening, the number of hidden neurons should increase. In this stage, the use of cross-validation technique and the analysis of validation and learning curves lead to a better understanding of how the hyper-parameters affect the neural network's performance. In addiction, for both models, lbfgs as optimization algorithm and relu as activation function produces the highest R^2. This work also shows that the usage of small neural networks capable of reproducing the material constitutive model is a plus to the continuous development in this area, due to the lower computational resources needed.

To test the artificial neural networks, mechanical tests outside the scope of the training were used. For both 1D and 2D models, the predicted curves of reverse cyclic loading fit well the virtual experimental results. However, the ANN-model (2D) presented residual values in components of total stress, back stress and in viscoplastic strain tensors, when they should be null.

The neural network on a finite element analysis software was successfully implemented. This implementation consists on a file method communication between a user-subroutine and the neural network model. On this stage, the predictions performed by the artificial neural network model can be compared under different integration schemes and increment sizes. The proposed model presented differences between them, however, both have a good performance when compared to the virtually experimental curves.

Knowing that the implementation is validated, the artificial neural network was tested for a more complex engineer problem. A tensile test along Y axis was performed using the adapted butterfly test specimen. Although the points used for analysis present combinations of strain never used for training, the neural network predicted curves similar to the ones produced by the explicit constitutive model.

The communication method implemented leads to a large wall-clock time, thus the creation of subroutine inside the UMAT describing the mathematical formulation of the trained neural network is proposed, as future work.

This work aims at creating a machine learning model to reproduce the explicit constitutive

model of a virtual material. Thus, the development of such model using experimental data is proposed as future work.

References

[1] A A Javadi, T P Tan, and M Zhang. "Neural network for constitutive modelling in finite element analysis". In: *Computer Assisted Mechanics and Engineering Sciences* 10.4 (2003), pp. 523–529.

[2] M. Lefik and B. A. Schrefler. "Artificial neural network as an incremental non-linear constitutive model for a finite element code". In: *Computer Methods in Applied Mechanics and Engineering* 192.28-30 (2003), pp. 3265–3283.

[3] Hou Man and Tomonari Furukawa. "Neural network constitutive modelling for non-linear characterization of anisotropic materials". In: *International Journal for Numerical Methods in Engineering* 85.August (2010), pp. 939–957.

[4] Tomonari Furukawa and Genki Yagawa. "Implicit constitutive modelling for viscoplasticity using neural networks". In: *International Journal for Numerical Methods in Engineering* 43.2 (1998), pp. 195–219.

[5] Y. M. A. Hashash, S. Jung, and J. Ghaboussi. "Numerical implementation of a neural network based material model in finite element analysis". In: *International Journal for Numerical Methods in Engineering* 59.7 (2004), pp. 989–1005.

[6] Jingwei Zhao, Hua Ding, Wenjuan Zhao, Mingli Huang, Dongbin Wei, and Zhengyi Jiang. "Modelling of the hot deformation behaviour of a titanium alloy using constitutive equations and artificial neural network". In: *Computational Materials Science* 92 (2014), pp. 47–56.

[7] Dong Dong Chen, Y. C. Lin, Ying Zhou, Ming Song Chen, and Dong Xu Wen. "Dislocation substructures evolution and an adaptive-network-based fuzzy inference system model for constitutive behavior of a Ni-based superalloy during hot deformation". In: *Journal of Alloys and Compounds* 708 (2017), pp. 938–946.

[8] Si Wei Wu, Xiao Guang Zhou, Guang Ming Cao, Zhen Yu Liu, and Guo Dong Wang. "The improvement on constitutive modeling of Nb-Ti micro alloyed steel by using intelligent algorithms". In: *Materials and Design* 116 (2017), pp. 676–685.

[9] Shuang Wu, Shougen Zhao, Dafang Wu, and Yunfeng Wang. "Constitutive modelling for restrained recovery of shape memory alloys based on artificial neural network". In: *NeuroQuantology* 16.5 (2018), pp. 806–813.

[10] Y. C. Lin, Fu Qi Nong, Xiao Min Chen, Dong Dong Chen, and Ming Song Chen. "Microstructural evolution and constitutive models to predict hot deformation behaviors of a nickel-based superalloy". In: *Vacuum* 137 (2017), pp. 104–114.

[11] Ali Nassr, Akbar Javadi, and Asaad Faramarzi. "Developing constitutive models from EPR-based self-learning finite element analysis". In: *International Journal for Numerical and Analytical Methods in Geomechanics* 42.3 (2017), pp. 401–417.

[12] J. Y. Ang, M. S. Abdul Majid, A. Mohd Nor, S. Yaacob, and M. J.M. Ridzuan. "First-ply failure prediction of glass/epoxy composite pipes using an artificial neural network model". In: *Composite Structures* 200 (2018), pp. 579–588.

[13] Eissa Fathalla, Yasushi Tanaka, and Koichi Maekawa. "Remaining fatigue life assessment of in-service road bridge decks based upon artificial neural networks". In: *Engineering Structures* 171.February (2018), pp. 602–616.

[14] Ankita Mangal and Elizabeth A. Holm. "Applied machine learning to predict stress hotspots I: Face centered cubic materials". In: *International Journal of Plasticity* 111 (2018), pp. 122–134.

[15] Mohammed S Kabbani and Hany A El Kadi. "Predicting the effect of cooling rate on the mechanical properties of glass fiber–polypropylene composites using artificial neural networks". In: *Journal of Thermoplastic Composite Materials* (2018), pp. 1–14.

[16] Anish Walia. *Activation functions and it's types-Which is better?* 2017. URL: `https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f` (visited on 03/22/2018).

[17] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: (2016), pp. 1–14.

[18] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer. "A Stochastic Quasi-Newton Method for Large-Scale Optimization". In: *SIAM Journal on Optimization* 26.2 (2014), pp. 1008–1031.

[19] Aria D. Haghighi. *Numerical Optimization: Understanding L-BFGS*. 2014. URL: `http://aria42.com/blog/2014/12/understanding-lbfgs` (visited on 03/22/2018).

[20] Shams S. *Neural Networks: Cost Function and Backpropagation — Machine Learning Medium*. 2018. URL: `https://machinelearningmedium.com/2017/10/03/neural-networks-cost-function-and-back-propagation/` (visited on 03/22/2018).

[21] Andrew Tchircoff. *The mostly complete chart of Neural Networks, explained.* URL: `https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464` (visited on 04/10/2018).

[22] Nimesh Sinha. *Understanding LSTM and its quick implementation in keras for sentiment analysis.* URL: `https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47` (visited on 04/24/2018).

[23] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2016. URL: `https://tensorflow.org/` (visited on 04/26/2018).

[24] Adam Paszke, Gregory Chanan, Zeming Lin, Sam Gross, Edward Yang, Luca Antiga, and Zachary Devito. "Automatic differentiation in PyTorch". In: *31st Conference on Neural Information Processing Systems* (2017), pp. 1–4.

[25] François Chollet. *Keras*. 2015. URL: `https://keras.io/` (visited on 04/26/2018).

[26] Fabian Pedregosa, Ron Weiss, and Matthieu Brucher. "Scikit-learn Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[27] Scikit-Learn. *sklearn.preprocessing.StandardScaler — scikit-learn 0.20.1 documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (visited on 05/15/2018).

[28] Scikit-Learn. *sklearn.metrics.mean_squared_error — scikit-learn 0.20.0 documentation*. URL: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html (visited on 05/18/2018).

[29] Scikit-Learn. *sklearn.metrics.r2_score — scikit-learn 0.20.0 documentation*. URL: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html (visited on 05/18/2018).

[30] Yurii Shevchuk. *Hyperparameter optimization for Neural Networks — NeuPy*. URL: http://neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html (visited on 05/25/2018).

[31] Scikit-Learn. *3.2. Tuning the hyper-parameters of an estimator — scikit-learn 0.20.0 documentation*. URL: http://scikit-learn.org/stable/modules/grid_search.html (visited on 06/02/2018).

[32] Scikit-Learn. *sklearn.model_selection.validation_curve — scikit-learn 0.20.0 documentation*. URL: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.validation_curve.html (visited on 06/08/2018).

[33] Adi Bronshtein. *Train/Test Split and Cross Validation in Python – Towards Data Science*. 2017. URL: https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6 (visited on 06/15/2018).

[34] Scikit-Learn. *3.5. Validation curves: plotting scores to evaluate models — scikit-learn 0.20.1 documentation*. URL: https://scikit-learn.org/stable/modules/learning_curve.html (visited on 06/08/2018).

[35] Joblib developers. *Persistence — joblib 0.13.0 documentation*. 2018. URL: https://joblib.readthedocs.io/en/latest/persistence.html (visited on 06/10/2018).

[36] Scipy. *scipy.integrate.odeint — SciPy v1.1.0 Reference Guide*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html (visited on 04/05/2018).

[37] Michael Smith. *ABAQUS/Standard User's Manual, Version 6.14*. Vol. IV. 2016.

[38] Imechanica.org. *User Subroutines in Abaqus*. 2017. URL: http://imechanica.org/files/WritingUserSubroutineswithABAQUS_0.pdf (visited on 06/25/2018).

[39] S. Thuillier J. Aquino, A. Andrade Campos, N. Souto. "Design of heterogeneous mechanical tests - Numerical methodology and experimental validation". In: *AIP Conference Proceedings* 1960.August (2018), pp. 1–20.

Appendices

Appendix A

Mechanical displacement used for the virtually experimental tests in 2D

A.1 Tensile test (T_{xx})

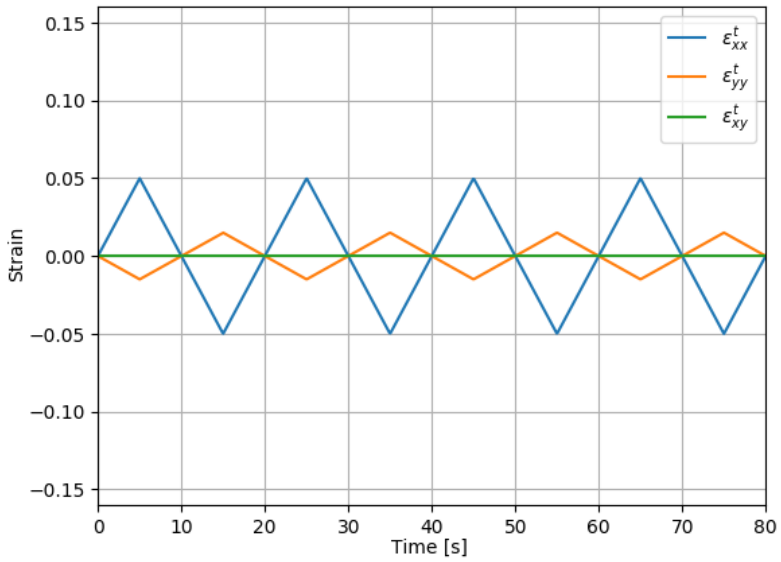


Figure A.1: Total strain tensor for a cyclic strain of ±0.05.

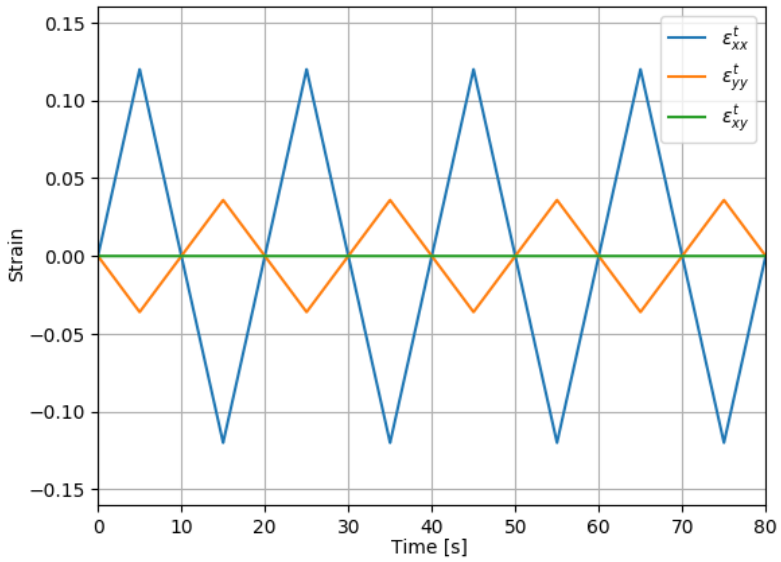


Figure A.2: Total strain tensor for a cyclic strain of ±0.12.

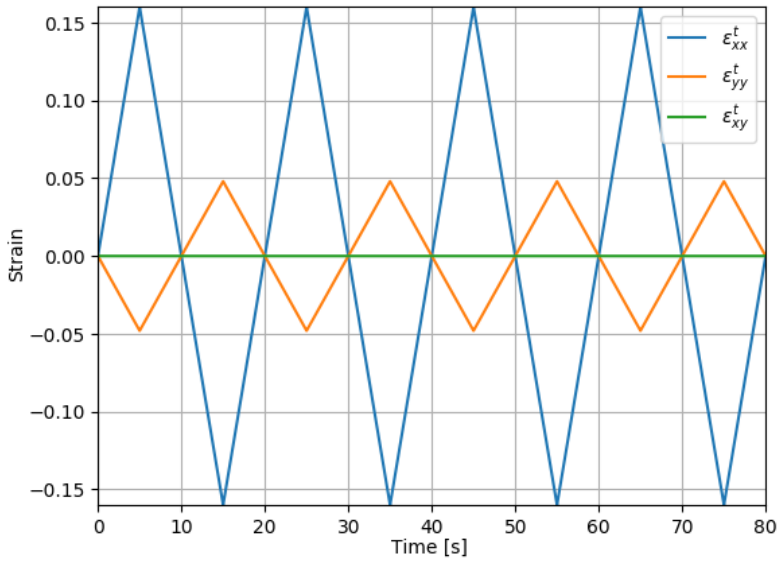


Figure A.3: Total strain tensor for a cyclic strain of ±0.16.

A.2 Tensile test (T_{yy})

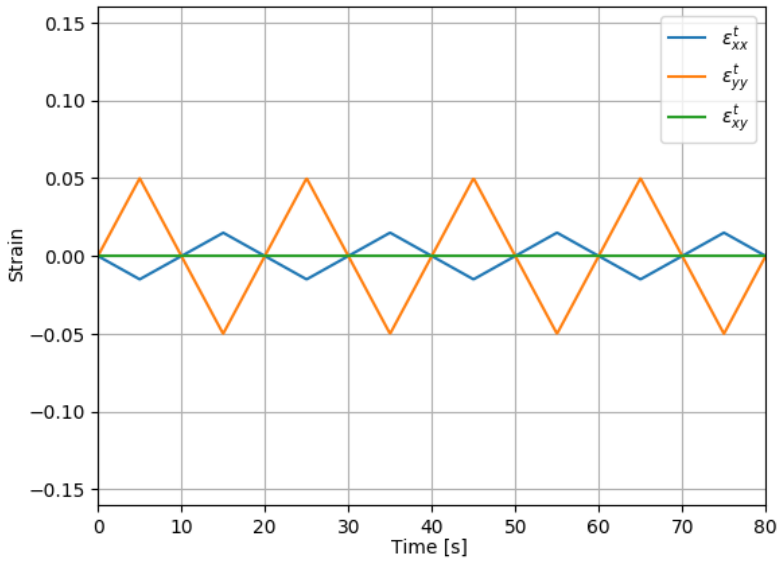


Figure A.4: Total strain tensor for a cyclic strain of ±0.05.

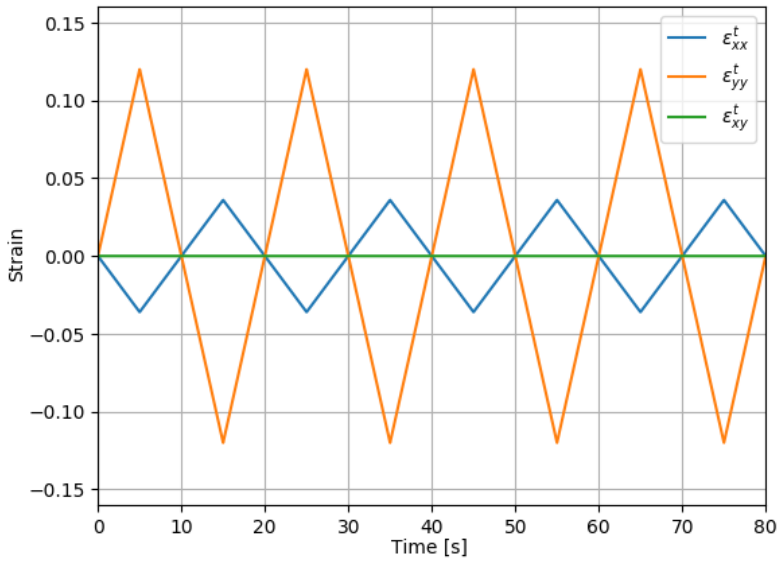


Figure A.5: Total strain tensor for a cyclic strain of ±0.12.

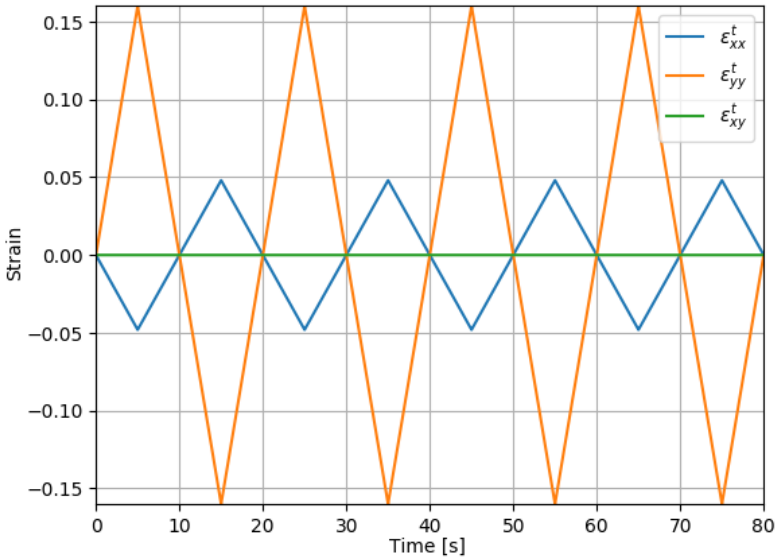


Figure A.6: Total strain tensor for a cyclic strain of ± 0.16.

A.3 Simple Shear test (S_{xy})

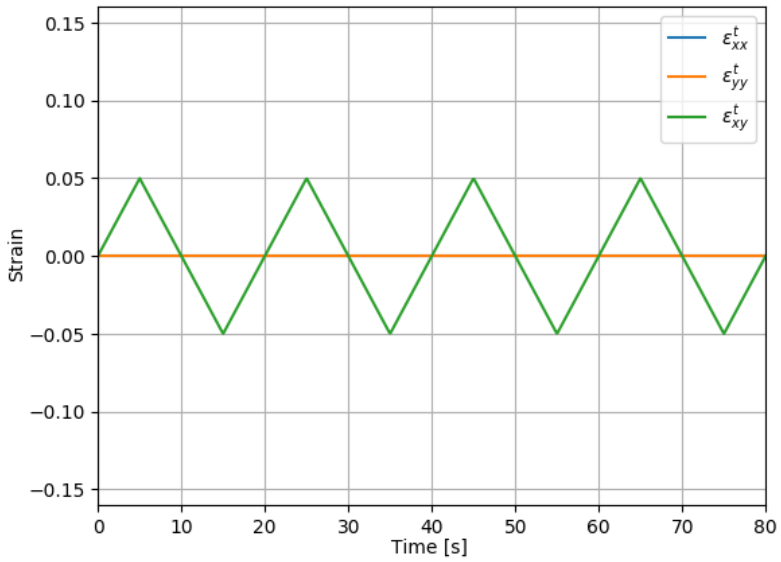


Figure A.7: Total strain tensor for a cyclic strain of ±0.05.

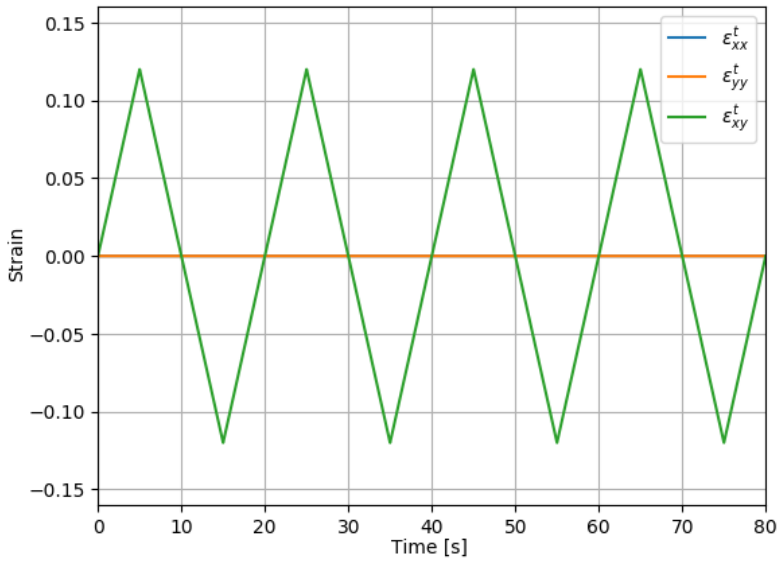


Figure A.8: Total strain tensor for a cyclic strain of ±0.12.

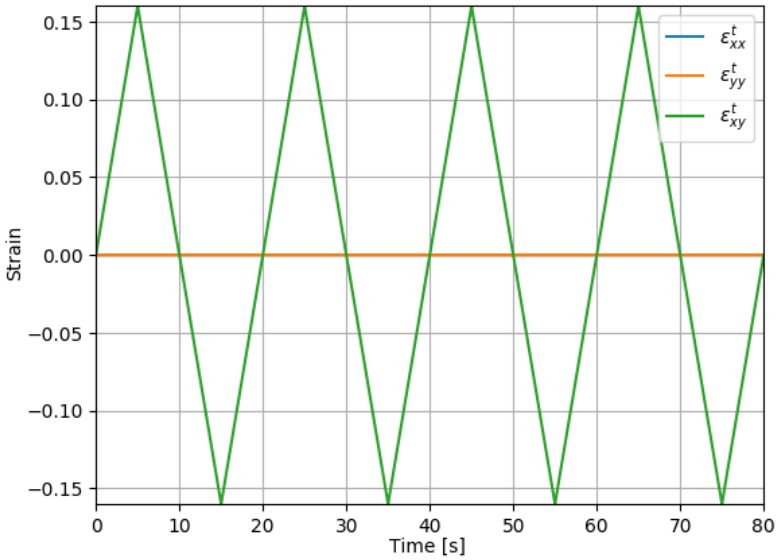


Figure A.9: Total strain tensor for a cyclic strain of ± 0.16.

Appendix B

Tables for validating ANN-model

Table B.1: Values calculated with ANN-model in comparison with the virtually experimental values in 2D. These values correspond at the maximum strain of the first cycle ($t = 5s$) on a strain range of ± 0.11.

Strain range			± 0.11		
Testing			T_{xx}	T_{yy}	S_{xy}
Total Stress [MPa]	σ_{xx}	experimental	293.22	0	0
		ANN-model	297.93	0	-10.38
	σ_{yy}	experimental	0	293.22	0
		ANN-model	0	293.80	15.46
	τ_{xy}	experimental	0	0	168.49
		ANN-model	4.35	-1.01	166.93
Back Stress [MPa]	χ_{xx}	experimental	97.38	-97.38	0
		ANN-model	92.91	-89.21	-21.27
	χ_{yy}	experimental	-97.38	97.38	0
		ANN-model	-94.47	90.63	20.97
	χ_{xy}	experimental	0	0	115.54
		ANN-model	5.87	3.48	97.28
Viscoplastic Strain [-]	$\varepsilon_{xx}^{\mathrm{vp}}$	experimental	6.67×10^{-2}	-6.67×10^{-2}	0
		ANN-model	6.33×10^{-2}	-6.85×10^{-2}	3.00×10^{-3}
	$\varepsilon_{yy}^{\mathrm{vp}}$	experimental	-6.67×10^{-2}	6.67×10^{-2}	0
		ANN-model	-5.84×10^{-2}	6.71×10^{-2}	-3.72×10^{-2}
	$\varepsilon_{xy}^{\mathrm{vp}}$	experimental	0	0	2.23×10^{-2}
		ANN-model	-2.26×10^{-3}	5.30×10^{-4}	2.32×10^{-2}
Plastic Strain [-]	p	experimental	7.70×10^{-2}	7.70×10^{-2}	1.82×10^{-2}
		ANN-model	7.81×10^{-2}	7.80×10^{-2}	2.95×10^{-2}
Drag Stress [MPa]	R	experimental	70.31	70.31	54.90
		ANN-model	70.50	68.81	57.95

Table B.2: Values calculated with ANN-model in comparison with the virtually experimental values in 2D. These values correspond at the maximum strain of the first cycle ($t = 5s$) on a strain range of ± 0.14.

Strain range			± 0.14		
Testing			T_{xx}	T_{yy}	S_{xy}
Total Stress [MPa]	σ_{xx}	experimental	306.77	0	0
		ANN-model	315.64	0	-3.90
	σ_{yy}	experimental	0	306.77	0
		ANN-model	0	310.25	20.95
	τ_{xy}	experimental	0	0	191.07
		ANN-model	5.56	0.81	194.42
Back Stress [MPa]	χ_{xx}	experimental	97.43	-97.43	0
		ANN-model	91.29	-87.21	-27.68
	χ_{yy}	experimental	-97.43	97.43	0
		ANN-model	-93.92	89.59	28.61
	χ_{xy}	experimental	0	0	132.77
		ANN-model	7.34	5.39	113.63
Viscoplastic Strain [-]	$\varepsilon_{xx}^{\mathrm{vp}}$	experimental	1.02×10^{-1}	-1.02×10^{-1}	0
		ANN-model	9.69×10^{-2}	-1.04×10^{-1}	2.04×10^{-3}
	$\varepsilon_{yy}^{\mathrm{vp}}$	experimental	-1.02×10^{-1}	1.02×10^{-1}	0
		ANN-model	-9.01×10^{-2}	1.02×10^{-1}	-4.42×10^{-3}
	$\varepsilon_{xy}^{\mathrm{vp}}$	experimental	0	0	4.06×10^{-2}
		ANN-model	-2.89×10^{-3}	-4.20×10^{-4}	3.88×10^{-2}
Plastic Strain [-]	p	experimental	1.18×10^{-1}	1.18×10^{-1}	3.31×10^{-2}
		ANN-model	1.19×10^{-1}	1.19×10^{-1}	4.93×10^{-2}
Drag Stress [MPa]	R	experimental	80.74	80.74	58.86
		ANN-model	80.84	77.63	63.15

Appendix C

Code

C.1 chaboche1D

```
#!/usr/bin/env python3
import math
import numpy as np
from scipy.integrate import odeint

class Chaboche1D:
    """


    Args:
        param1 (array): Array containing the material parameters.

    """


    def __init__(self, E, K, n, H, D, h, d):
        self.E = E
        self.K = K
        self.n = n
        self.H = H
        self.D = D
        self.h = h
        self.d = d
        self.solutions = []

    def total_strain(self, t):
        """

        Args:
            param1 (int): Current moment of time (t).

         Returns:
            The mechanical displacement for the current moment
            of time (t).
        """
        tc = 20.0                # Cyclic time for one cyclic loading
        Emax = 0.036             # Maximum mechanical displacement
        Emin = -Emax
```

```
        tcicle = t - tc*math.floor(t/tc)
        # Caculate total strain
        if tcicle <= tc/4.0:
            return 4.0*(Emax/tc)*tcicle
        if tc/4.0 < tcicle <= (3.0/4.0)*tc:
            return (-((4.0*Emax))/tc) * tcicle + (2.0) * Emax
        if (3.0/4.0)*tc < tcicle <= tc:
            return ((-4.0*Emin)/tc) * tcicle + 4.0*Emin


    def deriv(self, z, t, ET):
        """

        Args:
            param1 (np.array): Array containing the values of viscoplastic
                               strain, back stress and drag stress.
            param2 (np.array): A sequence of time points for which
                               to solve for z.
            param3 (int): The mechanical displacement for this step
        Returns:
            Array containing the derivatives of viscoplastic strain,
            back stress and drag stress in t, with the initial
            value z0 in the first row.

        """
        Evp = z[0]                      # Viscoplastic strain
        X = z[1]                        # Back stress
        R = z[2]                        # Drag stress
        S = self.E*(ET - Evp)           # Calculate Total Stress
        if abs(S - X) - R < 0:          # Elastic state
            dEvpdt = 0.
            dXdt = 0.
            dRdt = 0.
        else:                           # Plastic state
            dEvpdt = (((abs(S - X) - R) / self.K) ** self.n) * np.sign(S - X)
            dXdt = self.H * dEvpdt - self.D * X * abs(dEvpdt)
            dRdt = self.h * abs(dEvpdt) - self.d * R * abs(dEvpdt)
        return [dEvpdt, dXdt, dRdt]


    def solve(self, z0, t):
        """

        Args:
            param1 (np.array): Array containing the initial conditions.
            param2 (int): Total sequence of time points for which
                          to solve for z.

        """
        # Iterate through the sequence of time points
        for i in range(1, len(t)):
            # Mechanical displacement for next step
            ET = self.total_strain(t[i])
            # Time span for the next time step
            tspan = [t[i-1], t[i]]
            # Solve for next step
            z = odeint(self.deriv, z0, tspan, args=(ET,))
```

```
            # Store solution
            self.solutions.append(z)
            # Next initial condition
            z0 = z[1]

# Main program
if __name__ == "__main__":
    # initial conditions - Evp / X / R
    z0 = [0, 0, 50.0]
    # number of data points
    n = 3000
    # Define material parameters
    # E, K, n, H, D, h, d
    model_1D = Chaboche1D(5000.0, 50.0, 3.0, 5000.0, 100.0, 300.0, 0.6)
    # Time points
    t = np.linspace(0, 80, n)
    # Solve Chaboche's 1D model with given material parameters
    model_1D.solve(z0, t)
```

C.2 chaboche2D

```
#!/usr/bin/env python3
import math
import numpy as np
import copy
from scipy.integrate import odeint

class Chaboche2D:
    """


    Args:
        param1 (int): Array containing the material parameters,
                      as well as the maximum strain for the
                      specified test.
    Attributes:
        E,v,R1,k,K,a,b,c,n (int): Material parameters
        test (str): Type of mechanical test, it can be a tensile
                    test (xx,yy) or simple shear (xy).
        Emax (int): Maximum mechanical displacement

    """


    def __init__(self, E, v, R1, k, K, a, b, c, n, test, Emax):
        self.E = E
        self.v = v
        self.R1 = R1
        self.k = k
        self.K = K
        self.a = a
        self.b = b
        self.c = c
        self.n = n
        self.test = test
        self.Emax = Emax
        self.solutions = []
```

```
37
38      def total_strain(self, t):
39          """
40
41          Args:
42              param1 (int): Current moment of time (t).
43
44           Returns:
45              The mechanical displacement for the current moment
46              of time (t).
47          """
48          tc = 20.0                       # Cyclic time for one cyclic loading
49          Emax = self.Emax                # Maximum mechanical displacement
50          Emin = -Emax
51          tcicle = t - tc*math.floor(t/tc)
52          # Caculate total strain
53          if tcicle <= tc/4.0:
54              return 4.0*(Emax/tc)*tcicle
55          if tc/4.0 < tcicle <= (3.0/4.0)*tc:
56              return (-((4.0*Emax))/tc) * tcicle + (2.0) * Emax
57          if (3.0/4.0)*tc < tcicle <= tc:
58              return ((-4.0*Emin)/tc) * tcicle + 4.0*Emin
59
60
61      def deriv(self, z, t, stiff, ET):
62          """
63
64          Args:
65              param1 (np.array): Array containing the values of viscoplastic
66                                 strain tensor, back stress tensor, drag stress
67                                 and plastic strain.
68              param2 (np.array): A sequence of time points for which
69                                 to solve for z.
70              param3 (int): The mechanical displacement for this step
71          Returns:
72              Array containing the derivatives of viscoplastic strain tensor,
73              back stress tensor, drag stress and plastic strain in t, with
74              the initial value z0 in the first row.
75
76          """
77          Evp = z[:3].reshape(3, 1)       # Inelastic strain tensor
78          X = z[3:6].reshape(3, 1)        # Back stress tensor
79          R = copy.deepcopy(z[6])         # Drag stress
80          p = copy.deepcopy(z[7])         # plastic strain
81          ET = ET.reshape(3, 1)           # Total strain
82          # Calculate Stress
83          S = np.matmul(stiff, ET-Evp)
84          if self.test == 'xx':                   # Txx
85              S[1] = 0                            # S22 = 0
86          elif self.test == 'yy':                 # Txy
87              S[0] = 0                            # S11 = 0
88          # Calculate deviatoric Stress
89          S_dev = copy.deepcopy(S)
90          S_dev[0][0] -= (1./2.)*(S[0]+S[1])
91          S_dev[1][0] -= (1./2.)*(S[0]+S[1])
92          # Calculate deviatoric back stress
93          X_dev = copy.deepcopy(X)
```

```
        X_dev[0][0] -= (1./2.)*(X[0] + X[1])
        X_dev[1][0] -= (1./2.)*(X[0] + X[1])
        # Calculate J invariant
        J = math.sqrt((3./2.)*np.matmul((S_dev-X_dev).transpose(), S_dev-X_dev))

        if (J/self.K) < ((R + self.k)/self.K):          # Elastic State
            dpdt = 0
            dEvpdt = np.array([[0], [0], [0]])
            dXdt = np.array([[0], [0], [0]])
            dRdt = 0
        else:                                           # Plastic State
            # Calculate plastic strain rate
            dpdt = ((J - R - self.k) / self.K) ** self.n
            # Calculate viscoplastic strain rate tensor
            dEvpdt = (3./2.) * dpdt * (S_dev-X_dev)/J
            # Calculate Back stress rate tensor
            dXdt = (3./2.) * self.a * dEvpdt - self.c * X * dpdt
            # Calculate Drag stress rate
            dRdt = self.b * (self.R1 - R) * dpdt

        dzdt = [dEvpdt[0][0], dEvpdt[1][0], dEvpdt[2][0], dXdt[0, 0], dXdt[1, 0],
                dXdt[2, 0], dRdt, dpdt]
        return dzdt

    def solve(self, z0, t):
        """

        Args:
            param1 (np.array): Array containing the initial conditions.
            param2 (int): Total sequence of time points for which
                          to solve for z.

        """
        # Define Stiff matrix
        stiff = self.E/(1-self.v**2) * np.array([[1,       self.v,         0    ],
                                                 [self.v,    1 ,            0    ],
                                                 [0,         0,   (1-self.v)/2]])
        # Initialize Total strain tensor
        ET = np.zeros((1, 3))
        # Iterate through the sequence of time points
        for i in range(1, len(t)):
            # Mechanical displacements for next step
            if self.test == 'xx':                           # Txx test
                ET[0, 0] = self.total_strain(t[i])
                ET[0, 1] = -self.v * ET[0, 0]
            elif self.test == 'yy':                         # Tyy test
                ET[0, 1] = self.total_strain(t[i])
                ET[0, 0] = -self.v * ET[0, 1]
            elif self.test == 'xy':                         # Sxy test
                ET[0, 0] = 0
                ET[0, 1] = 0
                ET[0, 2] = self.total_strain(t[i])
            # Time span for next time step
            tspan = [t[i-1], t[i]]
            # Solve for next step
            z = odeint(self.deriv, z0, tspan, args=(stiff, ET))
            # store solution for plotting
```

```
            self.solutions.append(z)
            # Next initial condition
            z0 = z[1]

# Main program
if __name__ == "__main__":
    # initial conditions - Evp(tensor) / X(tensor) / R / p
    z0 = [0, 0, 0, 0, 0, 0, 50.0, 0]
    # number of data points
    n = 1000
    # Choose one test from -> (xx,yy,xy)
    test = 'xx'
    # Maximum mechanical displacement for cyclic loading
    Emax = 0.18
    # Define material and test parameters
    # E, v, R1, k, K, a, b, c, n, test, Emax
    model_2D = Chaboche2D(5000.0, 0.3, 500.0, 0.0, 50.0, 7500.0,
                          0.6, 100.0, 3.0, test, Emax)
    # Time points
    t = np.linspace(0, 80, n)
    # Solve Chaboche's 1D model with given material parameters
    model_2D.solve(z0, t)
```

C.3 UMAT

```
************************************************************************
**  UMAT FOR ABAQUS/STANDARD INCORPORATING PREDICTIVE MECHANICAL BEHAVIOUR **
**  BY NEURAL NETWORKS FOR PLANE-STRESS                                   **
************************************************************************
************************************************************************
**
**
**
*USER SUBROUTINE
      SUBROUTINE UMAT(STRESS,STATEV,DDSDDE,SSE,SPD,SCD,
     1 RPL,DDSDDT,DRPLDE,DRPLDT,
     2 STRAN,DSTRAN,TIME,DTIME,TEMP,DTEMP,PREDEF,DPRED,CMNAME,
     3 NDI,NSHR,NTENS,NSTATV,PROPS,NPROPS,COORDS,DROT,PNEWDT,
     4 CELENT,DFGRD0,DFGRD1,NOEL,NPT,LAYER,KSPT,KSTEP,KINC)
C
      INCLUDE 'ABA_PARAM.INC'
C
      CHARACTER*80 CMNAME
C
C
      DIMENSION STRESS(NTENS),STATEV(NSTATV),
     1 DDSDDE(NTENS,NTENS),DDSDDT(NTENS),DRPLDE(NTENS),
     2 STRAN(NTENS),DSTRAN(NTENS),TIME(2),DTIME(1),PREDEF(1),DPRED(1),
     3 PROPS(NPROPS),COORDS(3),DROT(3,3),DFGRD0(3,3),DFGRD1(3,3),
     4 KINC(1), KSTEP(1)
C
C
      PARAMETER (M=3,N=3,ID=3,ZERO=0.D0,ONE=1.D0,TWO=2.D0,THREE=3.D0,
     +           SIX=6.D0, NINE=9.D0, TOLER=0.D-6)
C
      DIMENSION DSTRESS(4), DDS(4,4), SDEV(3), XDEV(3)
      real(8) dEvp11, dEvp22, dEvp12, dR, dX11, dX22, dX12, dp
```

```
C
C     LOAD MATERIAL PROPERTIES AND STATE VARIABLES
C
      v = PROPS(1)
      E = PROPS(2)
      Evp11 = STATEV(1)
      Evp22 = STATEV(2)
      Evp12 = STATEV(3)
      X11 = STATEV(4)
      X22 = STATEV(5)
      X12 = STATEV(6)
      R = STATEV(7)
      p = STATEV(8)
C
C    CALCULATE ELASTIC STIFFNESS
C
      DDSDDE(1,1) = E/(1-v**2)
      DDSDDE(1,2) = (E*v)/(1-v**2)
      DDSDDE(1,3) = 0
      DDSDDE(2,1) = (E*v)/(1-v**2)
      DDSDDE(2,2) = E/(1-v**2)
      DDSDDE(2,3) = 0
      DDSDDE(3,1) = 0
      DDSDDE(3,2) = 0
      DDSDDE(3,3) = ((1-v)*E)/(2*(1-v**2))
C
C     ASSIGN STRESS TENSOR TO LOCAL VARIABLES
C
      S11 = STRESS(1)
      S22 = STRESS(2)
      S12 = STRESS(3)
C
C     ASK FOR THE DERIVATIVES
C
      a = get_derivatives(Evp11, Evp22, Evp12, R, S11, S22,
     + S12, X11, X22, X12, p)
C
C     READ THE DERIVATIVES
C
      open(unit=17, file='/home/miguel/derivatives.txt')
      read (17,*) dEvp11, dEvp12, dEvp22, dR, dX11, dX12, dX22, dp
      CLOSE(17)
C
C     TRANSFORM VISCOPLASTIC STRAIN RATE INTO INCREMENT
C
      Evp11_inc = dEvp11 * DTIME(1)
      Evp22_inc = dEvp22 * DTIME(1)
      Evp12_inc = dEvp12 * DTIME(1)
C
C     UPDATE STRESS
C
      STRESS(1) = S11 + DDSDDE(1,1)*(DSTRAN(1)-Evp11_inc)+
     +            DDSDDE(1,2)*(DSTRAN(2)-Evp22_inc)
      STRESS(2) = S22 + DDSDDE(2,1)*(DSTRAN(1)-Evp11_inc)+
     +            DDSDDE(2,2)*(DSTRAN(2)-Evp22_inc)
      STRESS(3) = S12 + DDSDDE(3,3)*(DSTRAN(3)-Evp12_inc)
C
```

```
C      UPDATE STATE VARIABLES
C
       STATEV(1) = Evp11 + dEvp11*DTIME(1)
       STATEV(2) = Evp22 + dEvp22*DTIME(1)
       STATEV(3) = Evp12 + dEvp12*DTIME(1)
       STATEV(4) = X11 + dX11*DTIME(1)
       STATEV(5) = X22 + dX22*DTIME(1)
       STATEV(6) = X12 + dX12*DTIME(1)
       STATEV(7) = R + dR*DTIME(1)
       STATEV(8) = p + dp*DTIME(1)
       RETURN
       END
**
**********************************************************************
**                          UTILITY     FUNCTIONS                    **
**********************************************************************
**
**
**********************************************************************
**          PREDICT STATE DERIVATIVES GIVEN A CERTAIN STATE           **
**********************************************************************
       real(8) function  get_derivatives(Evp11, Evp22, Evp12, R, S11, S22,
      1 S12, X11, X22, X12, p)
C
       real(8) Evp11, Evp22, Evp12, R, X11, X22, X12, p, S11, S22, S12
C
C      WRITE S,Evp,X TENSORS p AND R TO features.txt FILE
C
       open(unit=16, file='/home/miguel/features.txt')
       WRITE(16,*)Evp11,',',Evp12,',',Evp22,',',R,
      +            ',',S11,',',S12,',',S22,
      +            ',',X11,',',X12,',',X22,',',p
       CLOSE(16)
C
C      CALL MACHINE LEARNING MODEL TO MAKE PREDICTIONS
C      THEN WAIT UNTIL IT FINISH EXECUTION
C
       call system("/usr/bin/python /home/miguel/call_neuronalfem.py")
       return
       end
**********************************************************************
**  SDVINI FOR ABAQUS/STANDARD                                      **
**                                                                  **
**********************************************************************
**********************************************************************
**
**
**
*USER SUBROUTINE
       SUBROUTINE SDVINI(STATEV,COORDS,NSTATV,NCRDS,NOEL,NPT,
      1 LAYER,KSPT)
C
       INCLUDE 'ABA_PARAM.INC'
C
       DIMENSION STATEV(NSTATV),COORDS(NCRDS)
C
C      ASSIGN INITIAL CONDITIONS TO STATE VARIABLES
```

```
C
      STATEV(1) = 0.
      STATEV(2) = 0.
      STATEV(3) = 0.
      STATEV(4) = 0.
      STATEV(5) = 0.
      STATEV(6) = 0.
      STATEV(7) = 50.
      STATEV(8) = 0.
      RETURN
      END
```

C.4 neuronalFem (ANN-FEA)

```
#!/usr/bin/env python3
import numpy as np
from sklearn.externals import joblib

class NeuronalFem:
    """


    Args:
        param1 (str): Local path directory to machine learning model and
                      scalers.

    Attributes:
        estimator (pickle obj): Estimator ready for prediction using the
                                multi-layer perceptron model (ANN).
        scaler_x (pickle obj): Contains the binary file, which has the methods
                               to be used for scaling features.
        scaler_y (pickle obj): Contains the binary file, which has the methods
                               to be used for scaling predictions.

    """

    def __init__(self, modeldir):

        self.estimator = joblib.load(modeldir + 'ann.pkl')
        self.scaler_x = joblib.load(modeldir + 'scaler_x.pkl')
        self.scaler_y = joblib.load(modeldir + 'scaler_y.pkl')


    def get_features(self, filepath):
        """

        Args:
            param1 (str): Local path directory to the file containing
                          features
            Returns:
                Array containing the features (total stress tensor,
                viscoplastic strain tensor, back stress tensor, drag stress
                and plastic strain.)
        """
        file = open(filepath, 'r')
        for line in file.readlines():
            features = line.rstrip().split(',')
```

```
        features = [float(i) for i in features]
        file.close()
        return [features]


    def save_predictions(self, output, filepath):
        """

        Args:
            param1 (np.array): Array with the output predictions given
                               by estimator (ANN).
            param2 (str): Local path directory to the file where
                          predictions are stored
        Returns:
            None.
        """
        file = open(filepath, 'wb')
        # Initialize predictions vector
        predictions = np.arange(8, dtype=float)
        for i in range(0,8):
            predictions[i] = output[0][i]
        np.savetxt(file, [predictions], fmt='%0.6f', delimiter=',')
        file.close()
        return None

# Main program
if __name__ == "__main__":
    # Machine learning model directory
    modeldir = '/home/miguel/Documents/tese/ViscoPlastic-ML/2D/train/model/'
    # Initialize neuronalfem class with trained model for further prediction
    # and scalers to transform the data.
    neuronalfem = NeuronalFem(modeldir)
    # Load features from features.txt file
    features = neuronalfem.get_features('/home/miguel/features.txt')
    # Transform features values to make predictions
    input = neuronalfem.scaler_x.transform(features)
    # Make predictions and transform the output
    output = neuronalfem.scaler_y.inverse_transform(
                (neuronalfem.estimator.predict(input)))
    # Save predictions
    neuronalfem.save_predictions(output,'/home/miguel/predictions.txt')
```