Reducing computational time for FEM post-processing through the use of feedforward neural networks

Martin Zlatić[*], **Marko Čanađija**[†]

[*] Faculty of Engineering
University of Rijeka
Rijeka, Croatia
e-mail: mzlatic@riteh.hr

[†] Faculty of Engineering
University of Rijeka
Rijeka, Croatia
e-mail: marko.canadija@riteh.hr

Key words: Machine Learning, FEM, Postprocessing

Abstract: *With the recent surge in neural network usage, machine learning libraries have become more convenient to use and implement. In this paper the possibility of using neural networks in order to faster process displacements obtained from finite element calculation and replace existing post-processing procedures is investigated. The method is implemented on 2D membrane finite elements for their relative simplicity. A speed up is observed in comparison to traditional methods of post-processing. Possible further applications of this method are also presented in this paper.*

1 INTRODUCTION

As the performance of central processing units (CPUs) and graphics processing units (GPUs) increased in the past decades, neural networks have gained momentum since they can be implemented easier than ever before. This has given rise to code such as TensorFlow [1] and wrappers around the code to make it easier to use such as Keras [2]. Finite element calculations are commonly used by engineers in a plethora of fields [3] and the two are recently being combined and commonly used in describing constitutive models, multiscale simulations and other fields [4–12].

This paper has been inspired mainly by the work of Jung et al. [11] where neural networks are used to generate the finite element strain-displacement matrix in order to construct the element stiffness matrix. In this paper the possibility of calculating stress directly from nodal displacements using neural networks is presented as well as the speed increase over the FEM software Abaqus' post-processing. All the necessary data will be generated from Abaqus and Python with Keras will be used to train and evaluate neural networks.

2 PROBLEM STATEMENT

The goal of this paper is to correctly model linear elasticity trained on 2D membrane elements and directly obtain stress results from nodal displacements. A stiffness matrix of any finite element is given in eq. 1, where B is the strain displacement matrix and C is the material matrix.

$$K = \int_V B^T C B \, dV \tag{1}$$

In the paper by Jung [11] the strain displacement matrix is generated with neural networks, while in Huang et. al [12] only the material behaviour is captured. In Eq. 2 u are the nodal displacements of an element. Thus the need for having separate networks for generating a

https://doi.org/10.4995/YIC2021.2021.12473

strain displacement or material matrix can be eliminated and help speed up the calculation. Obtaining stresses from finite element calculations the following expression is used:

$$\sigma = CBu \tag{2}$$

3 DATA GENERATION AND PREPARATION

The data for training the neural network was obtained through the software Abaqus. First, a case was prepared through a script where a simple 4 point plate meshed with 2D membrane elements (type M3D4) was constrained at one edge and loaded on a different edge, Fig.1. The coordinates x_i, y_i on the plate were chosen randomly between 0.5 and 1.5 metres in their respective quadrants, and the nodal loads on the edge are all equal and are randomly chosen from $F_x^n \in \{-3000, 3000\}$ N , and $F_y^n \in \{-3000, 3000\}$ N with the individual nodal force then being $F^n = F_x^n \cdot i + F_y^n \cdot j$. A structured mesh of element size 50 mm was used. The force varies from case to case, as well as the edge to which the load or the constraint is applied. The minimum force applied to an edge can be 84 kN, while the maximum force can be 254 kN. The load direction also varies from case to case. In total 800 plates were auto-generated for obtaining training data, in total around 800 000 training samples.

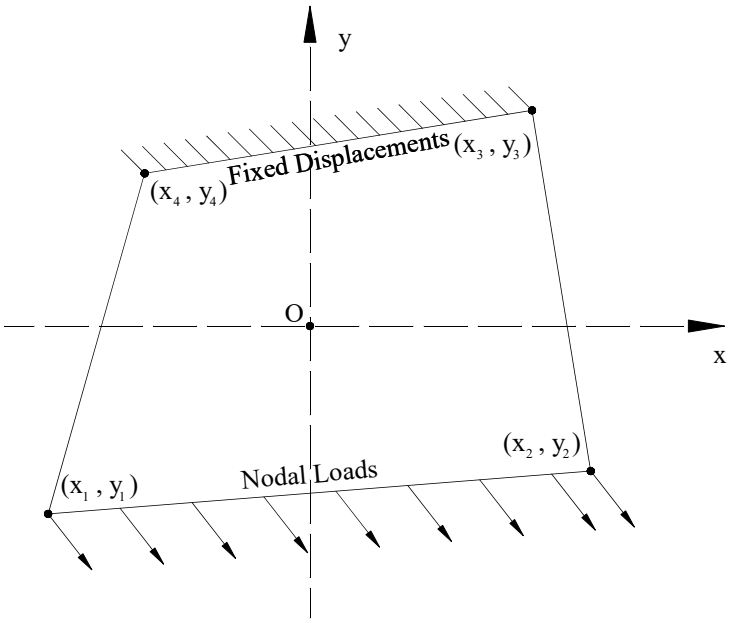

Figure 1: Plate geometry and boundary conditions for generating training data.

Once all the simulations are finished the data is processed in the following manner:

1. Nodal positions are obtained from the Abaqus input file
2. The intersection of the diagonals is found.
3. Distances between the intersection and nodes are found and stored in an auxiliary vector.
4. Displacements of the nodes are found and stored in an auxiliary vector.
5. Nodal positions and displacements and Poisson's ratio are added to the input vector for training.
6. Stress results from integration points are added into an output vector for training.

Data preparation is shown on Fig. 2. In total the input vector contains 17 values, nodal positions, displacements, and Poisson's ratio, see Eq. 3.

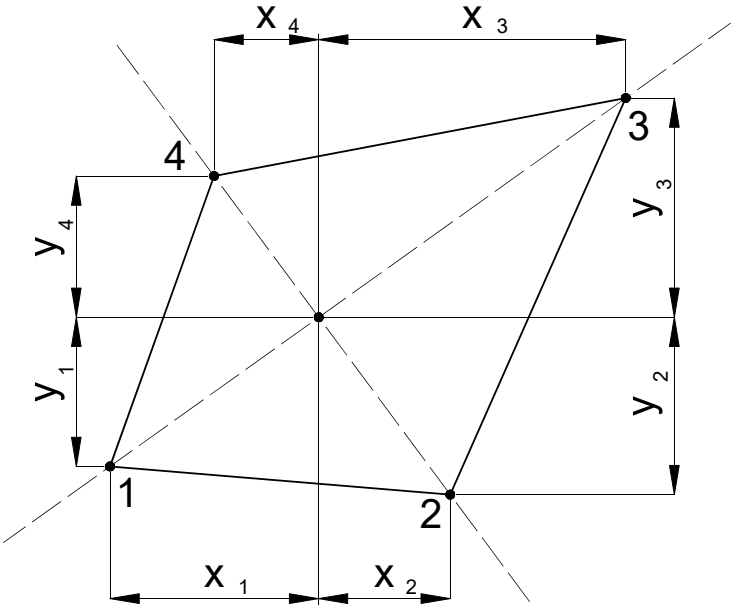

Figure 2: Illustration of diagonals intersection and nodal distances.

$$\boldsymbol{u} = (x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, u_x^1, u_y^1, u_x^2, u_y^2, u_x^3, u_y^3, u_x^4, u_y^4, \nu) \tag{3}$$

The output vector contains 12 values, 2 normal stresses and 1 shear stress per integration point, see Eq. 4. The lower indices refer to the stress component, and the upper indices refer to the integration point.

$$\boldsymbol{\sigma} = (\sigma_x^1, \sigma_y^1, \tau_{xy}^1, \sigma_x^2, \sigma_y^2, \tau_{xy}^2, \sigma_x^3, \sigma_y^3, \tau_{xy}^3, \sigma_x^4, \sigma_y^4, \tau_{xy}^4,) \tag{4}$$

4 TRAINING AND EVALUATING THE NETWORK

The hyperparameters of the network (number of layers, neurons per layer, activation function, kernel initialization) were determined through trial and error. An illustration of a general feed-forward neural network is given in Fig. 3 The best performing hyperparameters were:

- Number of hidden layers: 2
- Neurons per layer: 100
- Activation function: Parametric Rectified Linear Unit (PReLU)
- Kernel initialization: Glorot normal [13]
- Kernel regularizer: L2 regularization
- Bias: None
- Optimizer: Adam
- Loss measure: Mean squared error

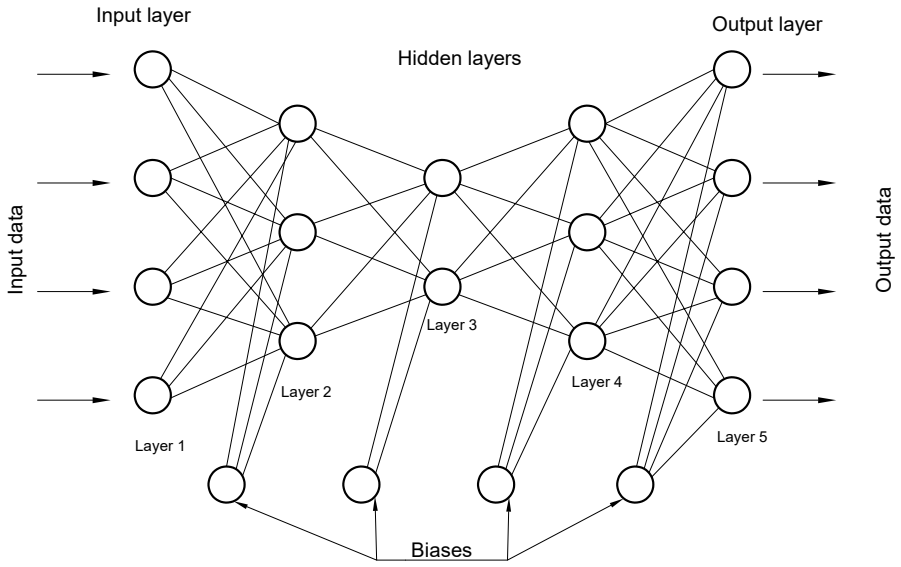

Figure 3: Illustration of a feed-forward neural network.

For each of the stress components a separate network was trained, due to this the output was subdivided into three outputs each consisting of 4 values.

$$\boldsymbol{\sigma_x} = (\sigma_x^1, \sigma_x^2, \sigma_x^3, \sigma_x^4) \tag{5}$$

$$\boldsymbol{\sigma_x} = (\sigma_y^1, \sigma_y^2, \sigma_y^3, \sigma_y^4) \tag{6}$$

$$\boldsymbol{\tau_{xy}} = (\tau_{xy}^1, \tau_{xy}^2, \tau_{xy}^3, \tau_{xy}^4) \tag{7}$$

The networks were then trained with early stopping enabled in case the validation loss does not improve for 10 epochs.

A few additional cases were generated that have not been in the training or validation set, these cases are declared to be the holdout set. As a general measure of accuracy the R^2 values are given in Table 1. The values were obtained on the holdout set.

Table 1: R^2 values for each stress component.

σ_x^1	σ_y^1	τ_{xy}^1	σ_x^2	σ_y^2	τ_{xy}^2	σ_x^3	σ_y^3	τ_{xy}^3	σ_x^4	σ_y^4	τ_{xy}^4
0.9906	0.9901	0.988	0.9915	0.975	0.978	0.9909	0.987	0.969	0.9905	0.975	0.969

A visual representation is given on Fig. 4. For brevity other plots like the one in Fig. 4 are not shown as they are very similar, as can be seen in Table 1.

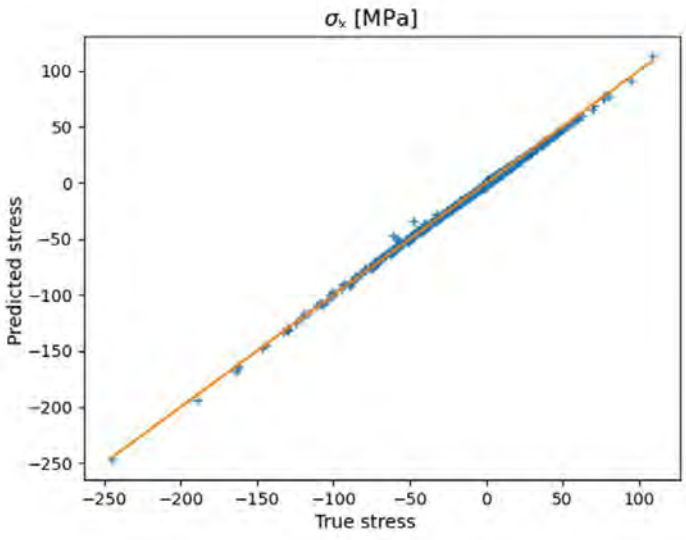

Figure 4: Plot of σ_x^1, predicted vs Abaqus stress.

5 TIME REDUCTION

Time was measured for Abaqus needed to complete the analysis of a plate with a set number of elements. Afterwards the simulation was rerun, but with the option to output stress results unchecked. The time difference between these two simulations is taken as the time necessary for Abaqus to post-process stress results. Then a dataset of the same size was processed by the previously obtained neural network and the execution time was measured. Given that 3 separate networks are used (one for each stress component) the execution time listed for the neural network is the total time for all 3 networks.

The time required for Abaqus to post-process stress results is 11 seconds while the execution time for the neural networks is 1.89 seconds. This translates into a time reduction of 82.8% or an acceleration of 5.82 times. Time required for saving the results from the networks to a file is also included in the neural network execution time (0.01 seconds per file save in NumPy).

6 CONCLUSION

Neural networks are a viable option for post-processing displacements of finite element calculations especially given the observed time reduction. In this paper they have been used in conjunction with 2D membrane finite elements and a linearly elastic material model. Applying neural networks to more complex material models such as those presented in Huang et al. [12] or du Bos et al. [9] and implementing them in non-linear solvers has the potential to reduce the computational time by a large margin.

Acknowledgments

This work has been fully supported by Croatian Science Foundation under the project IP-2019-04-4703. This support is gratefully acknowledged.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever,

K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[2] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[3] K. Bathe, *Finite element procedures*. Place of publication not identified: publisher not identified, 2006.

[4] J. Ghaboussi, D. A. Pecknold, M. Zhang, and R. M. Haj-Ali, "Autoprogressive training of neural network constitutive models," *International Journal for Numerical Methods in Engineering*, vol. 42, no. 1, pp. 105–126, may 1998.

[5] L. Liang, M. Liu, C. Martin, and W. Sun, "A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis," *Journal of The Royal Society Interface*, vol. 15, no. 138, p. 20170844, jan 2018.

[6] J. He, L. Li, J. Xu, and C. Zheng, "ReLU deep neural networks and linear finite elements," *Journal of Computational Mathematics*, vol. 38, no. 3, pp. 502–527, 2020.

[7] G. Capuano and J. J. Rimoli, "Smart finite elements: A novel machine learning application," *Computer Methods in Applied Mechanics and Engineering*, vol. 345, pp. 363–381, mar 2019.

[8] E. Haghighat, M. Raissi, A. Moure, H. Gomez, and R. Juanes, "A deep learning framework for solution and discovery in solid mechanics," 2020.

[9] M. L. du Bos, F. Balabdaoui, and J. N. Heidenreich, "Modeling stress-strain curves with neural networks: a scalable alternative to the return mapping algorithm," *Computational Materials Science*, vol. 178, p. 109629, jun 2020.

[10] P. Carrara, L. D. Lorenzis, L. Stainier, and M. Ortiz, "Data-driven fracture mechanics," *Computer Methods in Applied Mechanics and Engineering*, vol. 372, p. 113390, dec 2020.

[11] J. Jung, K. Yoon, and P.-S. Lee, "Deep learned finite elements," *Computer Methods in Applied Mechanics and Engineering*, vol. 372, p. 113401, dec 2020.

[12] D. Huang, J. N. Fuhg, C. Weißenfels, and P. Wriggers, "A machine learning based plasticity model using proper orthogonal decomposition," *Computer Methods in Applied Mechanics and Engineering*, vol. 365, p. 113008, jun 2020.

[13] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, 01 2010.