Accelerating finite element analysis using machine learning

Ghavamian, F.

DOI
10.4233/uuid:015bbf35-5e29-4630-b466-1a29d4c5bfb3

Publication date
2021

Document Version
Final published version

Citation (APA)
Ghavamian, F. (2021). *Accelerating finite element analysis using machine learning*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:015bbf35-5e29-4630-b466-1a29d4c5bfb3

ACCELERATING FINITE ELEMENT ANALYSIS USING MACHINE LEARNING

ACCELERATING FINITE ELEMENT ANALYSIS USING MACHINE LEARNING

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by the authority of the Rector Magnificus Prof. dr. ir. T. H. J. J. van der Hagen
chair of the Board for Doctorates,
to be defended publicly on Friday 17 September 2021 at 10:00 o'clock

by

Fariborz GHAVAMIAN

Master of Science in Civil Engineering,
Delft University of Technology, the Netherlands,
born in Tehran, Iran.

This dissertation has been approved by the promotor

Composition of the doctoral committee

Rector Magnificus,	voorzitter
Prof. dr. ir. L.J. Sluys,	Technische Universiteit Delft, promoter
Prof. dr. ir. A. Simone,	University of Padova, promoter
Independent members:	
Prof. E. Cueto	Universidad de Zaragoza
Dr. M.A. Bessa	Technische Universiteit Delft
Prof. dr. ir. L. Noels	Université de Liège
Prof. dr. ir. B.H.K. De Schutter	Technische Universiteit Delft
Other members:	
Dr. ir. P. Tiso	ETH Zürich
Reserve member:	
Prof. dr. ir. J.G. Rots,	Technische Universiteit Delft

The research presented in this thesis has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Grant agreement n° 617972.

Keywords: machine learning, deep learning, finite element analysis

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

To Sahar, without whom nothing matters much.

Contents

SUMMARY

We study the acceleration of the finite element method (FEM) simulations using machine learning (ML) models. Specifically, we replace computationally expensive (parts of) FEM models with efficient ML surrogates.

We develop three methods to speed up FEM simulations. The primary difference between these models is their degree of intrusion into the FEM source code. Here, we enumerate them from the most to the least intrusive.

In the first contribution, we tackle two bottlenecks of a FEM model equipped with a viscoplastic constitutive equation namely, solving the linear system of equations and evaluating the force vector. To tackle the former, we use a proper orthogonal decomposition (POD) method. And we tackle the latter with a discrete empirical interpolation method (DEIM). We observe that DEIM does not effectively speed up such a highly nonlinear FEM model. As a remedy, we divide the time domain into subdomains using a clustering algorithm. Then we construct a set of DEIM points for each cluster. By doing so, we manage to increase the efficiency of the POD-DEIM scheme. We, however, observe that the POD-DEIM scheme is sometimes unstable. The source of this instability is, to the extent of our knowledge, an open question.

In the second contribution, we consider a FE^2 scheme. The micro model is a FEM model equipped with a viscoplastic constitutive equation. The evaluation of the micro model is the computational bottleneck in this framework. Therefore, we develop a recurrent neural network as a surrogate for the micro model. In this contribution, we also propose a simple but effective sampling technique to collect stress-strain data points. The RNN model is trained based on this data. We also discuss how the RNN model becomes inaccurate when extrapolating. For these scenarios, we discuss how to improve the RNN by collecting more data and retraining.

In the third contribution, we develop a surrogate for the entire FEM simulation of a multi-physics problem. Specifically, we consider the FEM simulation of the electrochemical-mechanical interactions in a Li-ion battery. We propose a variant of the convolutional neural network (CNN), namely the HydraNet. The HydraNet takes the geometry of the battery and predicts all solution fields of the FEM model. Solution fields are either output of the solver or that of the post-processing. The HydraNet accepts inputs in the form of image-like fields. We discuss how to encode the geometry of the battery into a set of image-like fields.

We argue that the degree of intrusion of these methods to the FEM source code is inversely related to their industrial applicability. As a result, we believe that the first method (POD-DEIM) will mostly remain an academic contribution, while the other two could have potential industrial applications.

The central use case of these methods is in a multi-query application such as uncertainty quantification, design, and real-time simulations.

Samenvatting

We bestuderen de versnelling van de finite element method (FEM) simulaties met behulp van machine learning (ML) modellen. We vervangen vooral numeriek dure (delen van) FEM-modellen door efficiënte ML-surrogaten.

We ontwikkelen drie methoden om FEM-simulaties te versnellen. Het belangrijkste verschil tussen deze modellen is hun mate van ingrijpen in de FEM-broncode. Hier noemen we ze aflopend van de meest tot de minst ingrijpende.

In de eerste bijdrage pakken we twee knelpunten aan van een FEM-model uitgerust met een viscoplastische constitutieve vergelijking. Deze knelpunten zijn het oplossen van het lineare systeem van vergelijkingen en de evaluatie van de krachtenvector. Om het eerste aan te pakken, gebruiken we een proper orthogonal decomposition (POD) -methode. En het tweede knelpunt pakken we aan met een discrete empirical interpolation (DEIM). We stellen vast dat DEIM zo'n sterk niet-lineair FEM-model, niet effectief versnelt. Als remedie verdelen we het tijdsdomein in subdomeinen met behulp van een clusteralgoritme. Vervolgens construeren we een set DEIM-punten voor elk cluster. Daarmee slagen we erin om de efficiëntie van het POD-DEIM-schema te verhogen. We merken echter op dat het POD-DEIM-schema soms instabiel is. De bron van deze instabiliteit is, voor zover wij weten, een open vraag.

In de tweede bijdrage overwegen we een FE ^{2}-schema. In de tweede bijdrage beschouwen we een FE ^{2}-schema. Het micromodel is een FEM-model uitgerust met een viscoplastische constitutieve vergelijking. De evaluatie van het micromodel in dit verband, is het numeriek knelpunt. Daarom ontwikkelen we een recurrent neural network (RNN) als surrogaat voor het micromodel. In deze bijdrage stellen we ook een eenvoudige maar effectieve bemonsteringstechniek voor om spanning-rek datapunten te verzamelen. Op basis van deze data wordt het RNN-model getraind. We bespreken ook hoe het RNN-model onnauwkeurig wordt bij extrapolatie. Voor deze scenario's bespreken we hoe we de RNN kunnen verbeteren door meer data te verzamelen en opnieuw te trainen.

In de derde bijdrage ontwikkelen we een surrogaat voor de volledige FEM-simulatie van een multifysica probleem. We kijken specifiek naar de FEM-simulatie van de elektrochemische-mechanische interacties in een Li-ion-batterij. We stellen een variant van het convolutional neural network (CNN) voor, namelijk het HydraNet. Het HydraNet neemt de geometrie aan van de batterij en voorspelt alle oplossingsvelden van het FEM-model. Oplossingsvelden zijn ofwel het resultaat van de solver ofwel die van de nabewerking. Het HydraNet accepteert input in de vorm van image-like velden. We bespreken hoe de geometrie van de batterij in een reeks image-like velden kan worden gecodeerd.

Wij betogen dat de mate waarin deze methoden ingrijpen in de FEM-broncode omgekeerd gerelateerd is aan hun industriële toepasbaarheid. Als gevolg hiervan, denken we dat de eerste methode (POD-DEIM) vooral een academische bijdrage zal blijven, terwijl de andere twee potentiële industriële toepassingen zouden kunnen hebben.

In het algemeen worden deze methoden gebruikt in een multi-query-applicatie zoals onzekerheidskwantificering, ontwerp en real-time simulaties.

1

Introduction

A computer simulation is the process of mathematical modeling of a physical phenomenon in computers [1]. Physical phenomena are oftentimes modeled as a (set of) differential equations. The most widely used method to numerically solving these differential equations is the finite element method (FEM). For a brief history of FEM, we refer the reader to the inspiring talk of Strang [2].

Even though the application of FEM has been a success story, the yearing to use FEM models in multi-query applications is largely unsatisfied. The reason is that the speed of FEM simulations leaves much to be desired. In a multi-query application, one needs to execute the FEM model many times, and each time with a perturbation. Design, uncertainty quantification, and real-time simulation are examples of multi-query applications.

There are several classical methodologies to reduce the computational expense of FEM simulations. Let us briefly go through a number of them.

Solving the linear system of equations is a computational bottleneck in a FEM model. Advanced solvers, such as [3–5], significantly reduce this computational cost.

There are other contributions that aim at reducing the size of the system of equations through a generalized finite element method. Among many contribution, we refer to [6–8].

The evaluation of constitutive equations at each element is another computational bottleneck that is classically alleviated using parallel computing [9]. The central idea is to divide the domain of the problem into several subdomains and allocate a separate computer to each of them.

Even though the aforementioned methodologies manage to speed up FEM simulations, the speed gain is mostly not enough to enable multi-query applications.

In this study, we investigate another class of methodologies, namely machine learning (ML). An ML model is built based on sample data and oftentimes it is used for the purpose of interpolation. An interpolation problem is closely related to the approximation of computationally inefficient functions with efficient ones [10]. We make use of ML models in a similar fashion. Specifically, we investigate the possibility of using ML models as efficient surrogates for (parts of) FEM models. In the literature, this technique is also referred to as model order reduction (MOR).

In Chapter 2, we speed up the FEM simulation of viscoplastic deformation. To reduce the size of the system of equations, we employ the proper orthogonal decomposition method (POD). And to approximate the force vector we make use of the discrete empirical interpolation method (DEIM). We realize that DEIM cannot effectively speed up a highly nonlinear FEM model, such as the one equipped with a viscoplastic constitutive law. As a remedy, we divide the time domain of the FEM simulation into subdomains and we find a set of DEIM points for each one of them. In this manner, we manage to dramatically speed up the FEM simulation. Nevertheless, the POD-DEIM procedure remains unstable. This instability is a well-known phenomenon, but its source has not been concretely found. Nevertheless, several methods are proposed to alleviate it. Among these methods, we refer to choosing smarter DEIM points [11] and a different scheme to approximate the force vector namely, the energy-conserving sampling and weighting [12]. We, however, conjecture that POD and DEIM methods are not suited for highly nonlinear FEM models. The reason is that both POD and DEIM methods are based on the assumption that solution and force vectors can be defined in a lower-dimensional linear subspace. In the problem at hand, through observing the singular values of solution and force vectors, we realize that this assumption does not hold.

Additionally, the implementation of the POD-DEIM method requires deep intrusion into the FEM source code. One needs to modify the solver and the procedure that computes element contributions to the stiffness matrix and the force vector. The more intrusive a MOR technique is, the less appealing it is for an industrial application.

For these reasons, in chapters 3 and 4, we investigate less intrusive approaches that also use highly nonlinear ML models.

In chapter 3, we study accelerating a multi-scale FEM model. A multi-scale model consists of a macro and a micro model. The micro model serves as the constitutive equation of the macro model. For micro model we consider a FEM model equipped with a viscoplastic constitutive equation and it is executed at each integration point of the macro model. Incidentally, this is the main contributor to the computation expense of the multi-scale framework. To tackle this bottleneck, we develop a recurrent neural network (RNN) and use it as a surrogate for the micro model. Execution of the RNN model does not include computationally expensive operations such as the matrix inversion and the evaluation of constitutive equation. Therefore, in principle, it is considerably more efficient than a FEM model. At the same time, the RNN is a highly nonlinear ML model and can easily approximate the nonlinear behavior in a FEM simulation of viscoplastic deformation.

The scheme that we develop in chapter 3 is far less intrusive than POD-DEIM. To implement the RNN model, one needs to modify the interface between the macro and the micro models.

Development of the RNN however, requires more upfront investment than POD-DEIM. And that is due to the training cost of the RNN model.

In chapter 4, we investigate speeding up a multi-physics FEM model. Specifically, we consider a FEM model simulating electrochemical-mechanical interactions in a Li-ion battery. Our main purpose is to eliminate the need to intrude the FEM source code.

For that, we develop a variant of the convolutional neural network (CNN) that takes the geometry of the Li-ion battery and predicts all solution fields of the FEM model. Solution fields are outputs of the solver and the post-processing scheme. It is noteworthy to mention that the CNN model does not require the geometry of the battery to be described as mesh. It requires the geometry of the battery to be encoded into image-like fields. As a result, the computation cost due to mesh generation vanishes. We discuss how to encode the geometry of the battery into image-like fields.

The training cost of the HydraNet is also more than that of the POD-DEIM.

REFERENCES

[1] *Computer simulation,* https://en.wikipedia.org/wiki/Computer_simulation, accessed: 2020-12-26.

[2] G. Strang, *Finite element method - gilbert strang,* (2013), [Online; accessed December 2020].

[3] O. Schenk, K. Gärtner, W. Fichtner, and A. Stricker, *Pardiso: a high-performance serial and parallel sparse linear solver in semiconductor device simulation,* Future Generation Computer Systems **18**, 69 (2001), i. High Performance Numerical Methods and Applications. II. Performance Data Mining: Automated Diagnosis, Adaption, and Optimization.

[4] T. A. Davis, *Algorithm 832: Umfpack v4.3—an unsymmetric-pattern multifrontal method,* ACM Trans. Math. Softw. **30**, 196–199 (2004).

[5] P. R. Amestoy, I. S. Duff, and J.-Y. L'excellent, *Multifrontal parallel distributed symmetric and unsymmetric solvers,* Computer methods in applied mechanics and engineering **184**, 501 (2000).

[6] T. Strouboulis, I. Babuška, and K. Copps, *The design and analysis of the generalized finite element method,* Computer methods in applied mechanics and engineering **181**, 43 (2000).

[7] I. Babuška, U. Banerjee, and J. E. Osborn, *Generalized finite element methods—main ideas, results and perspective,* International Journal of Computational Methods **1**, 67 (2004).

[8] C. A. Duarte, I. Babuška, and J. T. Oden, *Generalized finite element methods for three-dimensional structural mechanics problems,* Computers & Structures **77**, 215 (2000).

[9] F. T. Mckenna, *Object-oriented finite element programming: Frameworks for analysis, algorithms and parallel computing.*, Doctoral dissertation, University of California, Berkeley (1999).

[10] *Interpolation,* https://en.wikipedia.org/wiki/Interpolation, accessed: 2020-12-27.

[11] B. Peherstorfer, Z. Drmac, and S. Gugercin, *Stability of discrete empirical interpolation and gappy proper orthogonal decomposition with randomized and deterministic sampling points,* SIAM Journal on Scientific Computing **42**, A2837 (2020).

[12] C. Farhat, T. Chapman, and P. Avery, *Structure-preserving, stability, and accuracy properties of the energy-conserving sampling and weighting method for the hyper reduction of nonlinear finite element dynamic models,* International Journal for Numerical Methods in Engineering **102**, 1077 (2015).

2

POD-DEIM MODEL ORDER REDUCTION FOR STRAIN SOFTENING VISCOPLASTICITY

Premature optimization is the root of all evil.

Donald Knuth

We demonstrate a Model Order Reduction technique for a system of nonlinear equations arising from the Finite Element Method (FEM) discretization of the three-dimensional quasistatic equilibrium equation equipped with a Perzyna viscoplasticity constitutive model. The procedure employs the Proper Orthogonal Decomposition-Galerkin (POD-G) in conjunction with the Discrete Empirical Interpolation Method (DEIM). For this purpose, we collect samples from a standard full order FEM analysis in the offline phase and cluster them using a novel k-means clustering algorithm. The POD and the DEIM algorithms are then employed to construct a corresponding reduced order model. In the online phase, a sample from the current state of the system is passed, at each time step, to a nearest neighbor classifier in which the cluster that best describes it is identified. The force vector and its derivative with respect to the displacement vector are approximated using DEIM, and the system of nonlinear equations is projected onto a lower dimensional subspace using the POD-G. The constructed reduced order model is applied to two typical solid mechanics problems showing strain localization (a tensile bar and a wall under compression) and a three-dimensional square-footing problem.

2.1. Introduction

In spite of constant advances in theoretical development and computing power, there are regularly applied many-query procedures, such as parameter sensitivity analysis or parameter estimation for inverse problems, that require many computationally expensive simulations. Model Order Reduction (MOR) techniques aim to reduce the computational cost of these procedures by breaking them into a two-phase procedure commonly

This chapter have been published in *Computer Methods in Applied Mechanics and Engineering* [1].

known as the offline-online decomposition. A reduced order model is first trained in the computationally expensive offline phase and then used in the online phase to efficiently carry out a many-query procedure. Successful examples can be found in structural dynamics [2], computational fluid dynamics [3, 4] and control systems [5]. Here we employ a set of machine learning techniques that can significantly improve the computational efficiency of a reduced order model.

The main hypothesis of any MOR technique can be put forth as follows [6]:

> "For a particular system, the solution space is often attracted to a low-dimensional manifold."

This hypothesis being true, it is in principle possible to build a set of basis vectors for a lower-dimensional subspace of the solution space, in which solution vectors can be approximated with acceptable accuracy, and therefore reduce the computational cost. The lower-dimensional subspace is built by means of samples collected from the solution space and the application of a Principal Component Analysis [7].

Many MOR techniques (Proper Orthogonal Decomposition-Galerkin (POD-G) [8], Balanced Truncation [9] and Moment Matching [10] to cite a few examples) have proven effective in reducing the computational cost of a linear model. However, they all fail to effectively reduce the computational cost of a nonlinear model [6], as the cost of the evaluation of the nonlinear contribution scales to the size of the full order system. To overcome this inefficiency, hyper-reduction methods, such as Missing Point Estimation [11], Gappy Proper Orthogonal Decomposition [12] and Discrete Empirical Interpolation Method (DEIM) [6], have been proposed to approximate the nonlinear contributions.

In the field of solid mechanics, MOR techniques such as POD-G [13, 14], Karhunen Loeve Expansion [15, 16], and Proper Generalized Decomposition (PGD) [17, 18] are employed to reduce the dimension of the system of equations. Nevertheless, the complexity of the evaluation of the constitutive equation remains a computational bottleneck. At variance with these models, in the hyper-reduction technique proposed in Reference [19] the integration of the constitutive equation is performed in a reduced integration domain. In Reference [20], an FE efficient unassembled variant of DEIM (UDEIM) is proposed and applied to geometrically nonlinear structural dynamics problems. In [21], UDEIM is further enhanced to reduce the associated offline cost. Apart from PGD, which is an a-priori method, these contributions focus only on the MOR technique itself, presuming that the underlying samples are fully capable of constructing an efficient reduced order model. We show that this is not necessarily always a valid assumption for the class of problems we are tackling here.

In this contribution we demonstrate an application of a MOR technique –POD-G, Section 2.3.1– in conjunction with a hyper-reduction method –DEIM, Section 2.3.2– to build a reduced order model for a boundary value problem –quasistatic equilibrium equation– with a nonlinear and path-dependent constitutive law –Perzyna viscoplasticity [22], Section 4.2. We observe that the singular values of the collected samples, which we wish had a fast decay as a proof of the central hypothesis of the MOR techniques, decrease at a slow rate. As a result, the efficiency of the reduced order model is diminished. To overcome this issue, Peherstorfer *et al.* [23] and Haasdonk *et al.* [24] divide the time domain and the parameter space into several local regions and construct a reduced order model for each one of them individually. Following a similar logic, we employ a set of Machine Learning techniques, such as the k-means clustering algorithm, Section 2.3.3, and the nearest neighbor classifier, Section 2.3.3, as suggested in [23]. As novel contributions, (i) we modify the distance definition in the k-means clustering algorithm to increase the consistency of the clustering procedure and, consequently, the effectiveness of DEIM; (ii) we modify the distance metric in the k-means clustering algorithm and the nearest neighbor classifier, which we heuristically show to be vital for a consistent classification; and (iii) we propose a trivial initial clustering in k-means clustering algorithm that exploits the path-dependent characteristic of the problem. The merits of the proposed MOR technique are demonstrated by three typical solid mechanics problems in Section 3.8.

2.2. PROBLEM STATEMENT

A strain-softening Perzyna viscoplasticity constitutive model is employed as an archetype of a class of constitutive models typically encountered in nonlinear solid mechanics. The three-dimensional quasistatic equilibrium equation

$$\boldsymbol{L}^{\mathrm{T}}\,\boldsymbol{\sigma} = \boldsymbol{q} \qquad \text{in} \quad \Omega, \tag{2.1}$$

expressed using matrix notation, is defined in the body Ω that is bounded by the surface $\Gamma = \Gamma_{\mathrm{u}} \cup \Gamma_{\mathrm{t}}$. A set of appropriate Dirichlet and Neumann boundary conditions is applied on Γ_{u} and Γ_{t}, respectively. In (2.1), $\boldsymbol{\sigma}$ is an array containing the components of the stress tensor in engineering notation, $\boldsymbol{q}$ the body force vector, $\boldsymbol{u}$ the displacement vector, and the operator matrix $\boldsymbol{L}$ is defined as

$$\boldsymbol{L}^{\mathrm{T}} = \begin{bmatrix} \partial_x & 0 & 0 & \partial_y & 0 & \partial_z \\ 0 & \partial_y & 0 & \partial_x & \partial_z & 0 \\ 0 & 0 & \partial_z & 0 & \partial_y & \partial_x \end{bmatrix}, \tag{2.2}$$

in which $\partial_{\#} = \frac{\partial}{\partial \#}$ is the partial derivative operator with respect to #. The equilibrium equation is equipped with the stress-strain relationship

$$\dot{\boldsymbol{\sigma}} = \boldsymbol{D}^{\mathrm{e}}\,(\dot{\boldsymbol{\epsilon}} - \dot{\boldsymbol{\epsilon}}^{\mathrm{vp}}) \tag{2.3}$$

in rate form, where $\dot{\#} = \frac{\partial \#}{\partial t}$ is the time derivative of #, $\boldsymbol{D}^{\mathrm{e}}$ represents the elastic modulus tensor, $\boldsymbol{\epsilon}$ represents the strain tensor, and $\boldsymbol{\epsilon}^{\mathrm{vp}}$ the viscoplastic strain tensor. In a Perzyna viscoplasticity constitutive model [22], the viscoplastic strain evolves following the flow rule

$$\dot{\boldsymbol{\epsilon}}^{\mathrm{vp}} = \dot{\lambda}\,\frac{\partial \theta}{\partial \boldsymbol{\sigma}} \tag{2.4}$$

with the rate of the plastic multiplier

$$\dot{\lambda} = \eta < \phi_{(\theta)} >^{\beta}, \tag{2.5}$$

the yield function

$$\theta = \sigma_{\mathrm{vm}} - \sigma_{\mathrm{Y}}, \tag{2.6}$$

and the overstress function

$$\phi_{(\theta)} = \frac{\theta}{\sigma_{\mathrm{Y}}}. \tag{2.7}$$

In the above equations, $< \# >$ is the Macaulay bracket, η is the viscosity parameter, β is a model parameter, σ_{Y} is the current yield stress, and σ_{vm} is the von Mises stress. To introduce a strain-softening response, we employ the relation

$$\sigma_{\mathrm{Y}} = \sigma_{\mathrm{Y}_0} \left((1+a)\, e^{-b\,\kappa} - a\, e^{-2b\,\kappa}\right), \tag{2.8}$$

where a and b are model parameters, σ_{Y_0} is the initial yield stress, and the plastic strain

$$\kappa = \lambda. \tag{2.9}$$

The strong form of the quasistatic equilibrium equation (2.1) is cast into its corresponding weak form and the displacement field is discretized in space using the Finite Element Method (FEM). Standard procedures yield the nonlinear discrete set of equations

$$\boldsymbol{r}(\boldsymbol{a}) = \boldsymbol{f}_{\mathrm{int}}(\boldsymbol{a}) - \boldsymbol{f}_{\mathrm{ext}} = \boldsymbol{0}, \tag{2.10}$$

where

$$\begin{aligned} \boldsymbol{f}_{\mathrm{int}}(\boldsymbol{a}) &= \int_{\Omega} \boldsymbol{B}^{\mathrm{T}} \boldsymbol{\sigma}(\boldsymbol{a})\, \mathrm{d}\Omega, \quad \text{and} \\ \boldsymbol{f}_{\mathrm{ext}} &= \int_{\Gamma_{\mathrm{t}}} \boldsymbol{N}^{\mathrm{T}} \bar{\boldsymbol{t}}\, \mathrm{d}\Gamma + \int_{\Omega} \boldsymbol{N}^{\mathrm{T}} \boldsymbol{q}\, \mathrm{d}\Omega. \end{aligned} \tag{2.11}$$

In (2.11), $\boldsymbol{N} \in \mathbb{R}^{3\times N}$ is a matrix that contains shape functions, $\boldsymbol{B} = \boldsymbol{L}\,\boldsymbol{N} \in \mathbb{R}^{6\times N}$ contains derivatives of the shape functions, $\bar{\boldsymbol{t}}$ are the prescribed tractions on Γ_{t}, $\boldsymbol{a} \in \mathbb{R}^{N}$ is a vector of the nodal values of the displacement vector $\boldsymbol{u}$, and N is the total number of degrees of freedom (DOFs).

The system of equations (2.10) is then solved using a Newton-Raphson scheme according to which

$$\begin{aligned} \boldsymbol{K}^{j}\, \mathrm{d}\boldsymbol{a}^{j+1} &= -\boldsymbol{r}^{j}, \qquad \text{and} \\ \boldsymbol{a}^{j+1} &= \boldsymbol{a}^{j} + \mathrm{d}\boldsymbol{a}^{j+1}, \end{aligned} \tag{2.12}$$

where $\boldsymbol{r}^{j} \in \mathbb{R}^{N}$ is the residual vector at iteration j, $\boldsymbol{K}^{j} \in \mathbb{R}^{N\times N}$ is the stiffness matrix at iteration j, and $\mathrm{d}\boldsymbol{a}^{j+1} \in \mathbb{R}^{N}$ is the increment of the displacement vector at iteration $j+1$. From here onward, the iteration number j is dropped for the sake of readability. The residual vector

$$\boldsymbol{r} = \boldsymbol{K}^{\mathrm{L}}\, \boldsymbol{a} + \boldsymbol{f} - \boldsymbol{f}_{\mathrm{ext}} \tag{2.13}$$

consists of a nonlinear part, $\boldsymbol{f}$, and a linear part, $\boldsymbol{K}^{\mathrm{L}}\, \boldsymbol{a} - \boldsymbol{f}_{\mathrm{ext}}$. In the linear part, the matrix

$$\boldsymbol{K}^{\mathrm{L}} = \int_{\Omega} \boldsymbol{B}^{\mathrm{T}}\, \boldsymbol{D}^{\mathrm{e}}\, \boldsymbol{B}\, \mathrm{d}\Omega \in \mathbb{R}^{N\times N} \tag{2.14}$$

is the linear elastic stiffness matrix, equal to the stiffness matrix $\boldsymbol{K}$ at the first iteration of the first time step. In the nonlinear part, the vector $\boldsymbol{f} = \boldsymbol{f}(\boldsymbol{a}) \in \mathbb{R}^{N}$ is referred to as the force vector. The derivative of the force vector, $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{a}} \in \mathbb{R}^{N\times N}$, generates the nonlinear part of the stiffness matrix.

Strain softening viscoplasticity typically induces strongly localized strain fields. As a result, a large number of degrees of freedom is required to resolve these gradients, thus causing two computational bottlenecks:

1. the solution of an $N \times N$ system (2.12.1), and
2. the evaluation of the force vector $\boldsymbol{f}$ and its derivative with respect to the displacement vector $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{a}}$.

2.3. Model Order Reduction

Essentially, a Model Order Reduction technique decomposes a many-query procedure into two parts: a computationally expensive offline phase, and a computationally efficient online phase. In the offline phase, a set of samples is collected from a standard analysis (in this context, this is done using FEM). This information is employed to construct a reduced order model. In the online phase, the reduced order model is used to carry out the many-query procedure. The MOR technique employed in this study is summarized in Section 2.3.5 while the corresponding online/offline procedures are detailed next.

The POD-G [6] is employed to project the nonlinear system of equations (2.10) onto a lower-dimensional subspace. DEIM [6] is then used to approximate the force vector and its derivative with respect to the displacement vector. In practice, the Proper Orthogonal Decomposition [25] is applied to a set of samples collected from a full order FEM analysis to compute a set of basis vectors. These basis vectors are later used in the POD-G and the DEIM procedures. Finally, the Localized Discrete Empirical Interpolation Method (LDEIM) [23] is employed to overcome a shortcoming of DEIM that occurs when the collected samples of the force vector are not attracted to a sufficiently lower-dimensional subspace. At variance with [23], we propose (i) a modified distance definition in the k-means clustering algorithm that is tailored for DEIM, (ii) a modified distance metric that is used in the k-means clustering algorithm and the nearest neighbor classifier, and (iii) a trivial initial clustering in the k-means clustering algorithm that exploits the path-dependent characteristic of the problem.

2.3.1. Proper Orthogonal Decomposition-Galerkin

Galerkin projection

To begin with, we attempt to project the nonlinear system of equations (2.10) onto a lower-dimensional subspace. Suppose that the solution of (2.10) is attracted to a lower-dimensional subspace $\mathscr{Y} \subset \mathbb{R}^N$. This solution can then be approximated by a weighted linear combination of the set of orthonormal basis vectors $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ ($k \ll N$) spanning $\mathscr{Y}$ as in

$$\boldsymbol{a} \approx \boldsymbol{V}\, \tilde{\boldsymbol{a}}, \tag{2.15}$$

where $\tilde{\boldsymbol{a}} \in \mathbb{R}^k$ is a set of scalar weights associated to the set of basis vectors $\boldsymbol{V} \in \mathbb{R}^{N\times k}$. Inserting (2.15) into (2.10) yields an overdetermined nonlinear system of N equations in k unknowns. To compute an approximate value for $\tilde{\boldsymbol{a}}$, we construct an equivalent system of equations $\boldsymbol{g} = \boldsymbol{0}$ that is determined. To find such a system, we search for a set of k functions $\boldsymbol{g}$ in subspace $\mathscr{Y}$ such that, for any given value of $\tilde{\boldsymbol{a}}$, the L_2 norm of the difference between the residual $\boldsymbol{r}(\boldsymbol{V}\tilde{\boldsymbol{a}})$ and its counterpart in the lower-dimensional subspace $\boldsymbol{V}\boldsymbol{g}$ is minimized according to [26, Theorem 3.2]

$$\arg\min_{\boldsymbol{g}} \| \boldsymbol{r}(\boldsymbol{V}\tilde{\boldsymbol{a}}) - \boldsymbol{V}\boldsymbol{g} \|_2. \tag{2.16}$$

Owing to the choice of the L_2 norm, it is straightforward to solve (2.16) for $\boldsymbol{g}$. Consequently, the equivalent system of equations $\boldsymbol{g} = \boldsymbol{0}$ becomes

$$\boldsymbol{V}^{\mathrm{T}} \boldsymbol{r}(\boldsymbol{V}\tilde{\boldsymbol{a}}) = \boldsymbol{0}. \tag{2.17}$$

The system of equations (2.17) is then solved using a Newton-Raphson scheme according to which

$$\begin{aligned} \tilde{\boldsymbol{K}}^j \, \mathrm{d}\tilde{\boldsymbol{a}}^{j+1} &= -\tilde{\boldsymbol{r}}^j, \qquad \text{and} \\ \tilde{\boldsymbol{a}}^{j+1} &= \tilde{\boldsymbol{a}}^j + \mathrm{d}\tilde{\boldsymbol{a}}^{j+1}, \end{aligned} \tag{2.18}$$

where $\tilde{\boldsymbol{K}}^j = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{K}^j \boldsymbol{V} \in \mathbb{R}^{k\times k}$ is the reduced stiffness matrix and $\tilde{\boldsymbol{r}}^j = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{r}^j \in \mathbb{R}^k$ is the reduced residual vector. The set of basis vectors $\boldsymbol{V}$ is computed using the Proper Orthogonal Decomposition as explained in Section 2.3.1.

In spite of the system of equations (2.18) being of dimension $k \ll N$, the complexity of problem still depends on N when computing the nonlinear contributions. Efficient reduction is achieved later in Sections 2.3.2 and 2.3.3 by approximating the force vector and its derivative with respect to the displacement vector using DEIM.

Proper Orthogonal Decomposition

A set of orthonormal basis vector $\{\boldsymbol{v}_1,\ldots,\boldsymbol{v}_k\}$ is used to project the system of equations (2.10) onto a lower-dimensional subspace as in (2.17). This set is computed by means of the Proper Orthogonal Decomposition (POD) method [25, 27]. For this purpose, samples of the displacement and the force vectors are collected from a standard full order FEM analysis and gathered in the sample matrices $\boldsymbol{A}^{\mathrm{samp}} = [\boldsymbol{a}_1,\ldots,\boldsymbol{a}_{n_s}]$ and $\boldsymbol{F}^{\mathrm{samp}} = [\boldsymbol{f}_1,\ldots,\boldsymbol{f}_{n_s}]$, respectively. For the displacement samples (and similarly for the force samples), we search for a subspace $\mathscr{Y} = \mathrm{span}\{\boldsymbol{v}_1,\ldots,\boldsymbol{v}_k\} \subset \mathbb{R}^N$ in which the samples can be optimally described. In other words, it is preferable that the error between samples and their projection on $\mathscr{Y}$ is minimized in a certain sense. Formally, the set of orthonormal basis vectors spanning $\mathscr{Y}$ is computed by solving the optimization problem [6]

$$\begin{aligned} &\arg\min_{\{\boldsymbol{v}_1,\ldots,\boldsymbol{v}_k\}} \sum_{i=1}^{n_s} \|\boldsymbol{a}_i - \sum_{j=1}^{k} (\boldsymbol{v}_j^{\mathrm{T}}\boldsymbol{a}_i)\, \boldsymbol{v}_j\|_2^2, \\ &\text{subjected to} \quad \boldsymbol{v}_m^{\mathrm{T}}\boldsymbol{v}_n = \begin{cases} 1 & m = n \\ 0 & m \neq n \end{cases}, \quad m,n = 1,\ldots,k. \end{aligned} \tag{2.19}$$

In the equation above, the error between a sample $\boldsymbol{a}_i$ and its projection $\sum_{j=1}^{k}(\boldsymbol{v}_j^{\mathrm{T}}\boldsymbol{a}_i)\, \boldsymbol{v}_j$ onto the subspace $\mathscr{Y}$ is minimized in an L_2 sense. Note that the norms in (2.19) are squared to facilitate the derivation of (2.21). It can be shown [27] that the solution to (2.19) is a set of left singular vectors $\boldsymbol{V}$ obtained from the application of the Singular Value Decomposition (SVD) to the sample matrix $\boldsymbol{A}^{\text{samp}}$ such that

$$\boldsymbol{V}\,\boldsymbol{\Sigma}\,\boldsymbol{W}^{\text{T}} = \boldsymbol{A}^{\text{samp}}, \tag{2.20}$$

where $\boldsymbol{V} \in \mathbb{R}^{N\times N}$ is a set of left orthonormal singular basis vectors, $\boldsymbol{W}^{n_s\times n_s}$ is a set of right orthonormal singular basis vectors, and $\boldsymbol{\Sigma}$ is a diagonal matrix that contains the singular values in a descending manner $\sigma_1 > ... > \sigma_r$, $r = \text{rank}\,(\boldsymbol{A}^{\text{samp}})$. Therefore, by using an SVD [27, Theorem 1.1.1], the residual of (2.19) becomes [6]

$$\sum_{i=1}^{n_s} \|\boldsymbol{a}_i - \sum_{j=1}^{k} (\boldsymbol{v}_j^{\text{T}}\boldsymbol{a}_i)\,\boldsymbol{v}_j\|_2^2 = \sum_{i=k+1}^{r} \sigma_i^2. \tag{2.21}$$

From the equation above, it is possible to infer that the projection error scales to the sum of the squares of the neglected ($i = k+1$ to r) singular values (RHS of (2.21)). As a result of this observation, a heuristic criterion to select the number of basis vectors k would be to consider the ratio of the sum of the squares of the neglected singular values to the sum of all the singular values as in

$$\frac{\sum_{i=k+1}^{r}\sigma_i^2}{\sum_{i=1}^{r}\sigma_i^2} \leq \epsilon \tag{2.22}$$

where $\epsilon \in [0\ 1]$ is a user-defined tolerance. As the value of ϵ deviates from one and tends to zero, the number of neglected singular values decreases. Simultaneously, the number of basis vectors which are taken into account increases, yielding a more accurate reduced order model. The POD procedure is summarized in Algorithm 1.

```
Algorithm 1: POD procedure
    input  : Samples $\boldsymbol{A}^{\text{samp}}$, a tolerance $\epsilon$
    output: Truncated basis vectors $\boldsymbol{V} \in \mathbb{R}^{N\times k}$

        Apply Singular Value Decomposition: $\boldsymbol{A}^{\text{samp}} = \boldsymbol{V}\,\boldsymbol{\Sigma}\,\boldsymbol{W}^{\text{T}}$
        Truncate $\boldsymbol{V}$ based on $\boldsymbol{\Sigma}$ and criterion (2.22)
```

2.3.2. Discrete Empirical Interpolation Method

We now tackle the computational bottleneck related to the complexity of the evaluation of the force vector and its derivative with respect to the displacement vector. This is done by employing DEIM [6]. Suppose that the force vector $\boldsymbol{f}$ in (2.13) is attracted to a lower-dimensional subspace $\mathscr{U} = \text{span}\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m\} \subset \mathbb{R}^N$, where $m \ll N$. The force vector can then be approximated using a weighted linear combination of the basis vectors spanning $\mathscr{U}$ according to

$$\boldsymbol{f} \approx \hat{\boldsymbol{f}} = \boldsymbol{U}\,\boldsymbol{c}, \tag{2.23}$$

where $\boldsymbol{c} \in \mathbb{R}^m$ is a set of scalar weights associated to the set of basis vectors $\boldsymbol{U}$. To find these scalar weights, m rows of both sides of (2.23) are selected by means of a matrix $\boldsymbol{P} \in \mathbb{R}^{N\times m}$:

$$\boldsymbol{P}^{\text{T}}\boldsymbol{f} \approx \boldsymbol{P}^{\text{T}}\boldsymbol{U}\,\boldsymbol{c}. \tag{2.24}$$

Solving (2.24) for $\boldsymbol{c}$ and inserting it in (2.23) yield the approximated force vector

$$\hat{\boldsymbol{f}} = \boldsymbol{U}\,(\,\boldsymbol{P}^{\mathrm{T}}\boldsymbol{U}\,)^{-1}\boldsymbol{P}^{\mathrm{T}}\boldsymbol{f} = \boldsymbol{U}\,(\,\boldsymbol{P}^{\mathrm{T}}\boldsymbol{U}\,)^{-1}\,\boldsymbol{f}_{\mathrm{DEIM}}. \tag{2.25}$$

Here, $\boldsymbol{f}_{\mathrm{DEIM}} = \boldsymbol{P}^{\mathrm{T}}\boldsymbol{f} \in \mathbb{R}^{m}$ contains m entries of the force vector, and the matrix $\boldsymbol{P}$ is defined with the aid of the DEIM algorithm listed in the next subsection. The complexity of the evaluation of $\boldsymbol{f}$ is now reduced to that of $\hat{\boldsymbol{f}}$; thus, this consists of the evaluation of $\boldsymbol{f}_{\mathrm{DEIM}}$ and a matrix multiplication (note that the computation of $\boldsymbol{U}\,(\,\boldsymbol{P}^{\mathrm{T}}\boldsymbol{U}\,)^{-1}$ is carried out in the offline phase).

To compute the derivative of the approximated force vector (2.25) with respect to the displacement vector, several methods are suggested in [28]. Here we choose to utilize the same set of basis vectors previously computed for the force vector, as originally suggested in [6]:

$$\frac{\partial \hat{\boldsymbol{f}}}{\partial \boldsymbol{V}\tilde{\boldsymbol{a}}} = \boldsymbol{U}\,(\,\boldsymbol{P}^{\mathrm{T}}\boldsymbol{U}\,)^{-1}\boldsymbol{P}^{\mathrm{T}}\,\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{V}\tilde{\boldsymbol{a}}} = \boldsymbol{U}\,(\,\boldsymbol{P}^{\mathrm{T}}\boldsymbol{U}\,)^{-1}\,\boldsymbol{K}_{\mathrm{DEIM}}, \tag{2.26}$$

where $\boldsymbol{K}_{\mathrm{DEIM}} = \boldsymbol{P}^{\mathrm{T}}\,\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{V}\tilde{\boldsymbol{a}}} \in \mathbb{R}^{m\times N}$ contains m rows of the nonlinear part of the stiffness matrix. The procedure to assemble $\boldsymbol{f}_{\mathrm{DEIM}}$ and $\boldsymbol{K}_{\mathrm{DEIM}}$ is detailed in Algorithm 2.

Algorithm 2: Assembly of $\boldsymbol{K}_{\mathrm{DEIM}}$ and $\boldsymbol{f}_{\mathrm{DEIM}}$

```
input  : A set $\boldsymbol{IND} = [IND_1,\ldots,IND_m]$ of entries selected by the DEIM
         Algorithm 3
output: $\boldsymbol{K}_{\mathrm{DEIM}}$ and $\boldsymbol{f}_{\mathrm{DEIM}}$

    Set $\boldsymbol{K}_{\mathrm{DEIM}}$ and $\boldsymbol{f}_{\mathrm{DEIM}}$ to zero
    for $i = 1,\ldots,m$ do
        Find elements $\boldsymbol{E} = \{E_1,\ldots,E_{n_e}\}$ sharing the $IND_i$ entry
        for $e = 1\ \ldots\ n_e$ do
            Compute the nonlinear part of element stiffness matrix and the force
              vector: $\boldsymbol{K}_e^{\mathrm{E}}$ and $\boldsymbol{f}_e^{\mathrm{E}}$
            Find the local DOF $j$ associated to the entry $IND_i$
            $\boldsymbol{f}_{\mathrm{DEIM}}(i) = \boldsymbol{f}_{\mathrm{DEIM}}(i) + \boldsymbol{f}_e^{\mathrm{E}}(j)$
            Define $\boldsymbol{gIndex}$ as the vector of global DOFs associated with element
              $e$
            $\boldsymbol{K}_{\mathrm{DEIM}}(i,\ \boldsymbol{gIndex}) = \boldsymbol{K}_{\mathrm{DEIM}}(i,\ \boldsymbol{gIndex}) + \boldsymbol{K}_e^{\mathrm{E}}(j,\ :)$
        end
    end
```

Discrete Empirical Interpolation Method algorithm

The error bound for the DEIM approximation $\hat{\boldsymbol{f}}$ of the force vector $\boldsymbol{f}$ is given by [6]

$$\|\boldsymbol{f} - \hat{\boldsymbol{f}}\|_2 \leq \|(\boldsymbol{P}^{\mathrm{T}}\boldsymbol{U})^{-1}\|_2\ \|(\boldsymbol{I} - \boldsymbol{U}\,\boldsymbol{U}^{\mathrm{T}})\boldsymbol{f}\|_2. \tag{2.27}$$

To increase the accuracy of the DEIM approximation, the value of $\|(\boldsymbol{I} - \boldsymbol{U}\,\boldsymbol{U}^{\mathrm{T}})\boldsymbol{f}\|_2$ should be decreased while the value of $\|(\boldsymbol{P}^{\mathrm{T}}\boldsymbol{U})^{-1}\|_2$ is kept bounded [6, Lemma 3.2].

The quantity $\|(\boldsymbol{I}-\boldsymbol{U}\,\boldsymbol{U}^{\mathrm{T}})\boldsymbol{f}\|_2$ is the projection error induced by POD (2.21). Therefore, by increasing the number of basis vectors in $\boldsymbol{U}$, according to (2.22), its value becomes smaller. However, by increasing the number of basis vectors, the value of $\|(\boldsymbol{P}^{\mathrm{T}}\boldsymbol{U})^{-1}\|_2$ becomes larger, jeopardizing the accuracy of the DEIM approximation. According to Chaturantabut and Sorensen [6, Lemma 3.2], the DEIM Algorithm 3 computes the matrix $\boldsymbol{P}$ in a way that the growth of this term is bounded. Consequently, the accuracy of the approximation can be improved by taking more and more basis vectors into account.

Algorithm 3: DEIM algorithm

```
input  : Basis vectors $\boldsymbol{U} = [\boldsymbol{u}_1, \dots, \boldsymbol{u}_m]$
output: Entries of the force vector selected by
        DEIM  $\boldsymbol{IND} = [IND_1, \dots, IND_m]$, and
        the matrix $\boldsymbol{P} = [\boldsymbol{e}_{IND_1}, \dots, \boldsymbol{e}_{IND_m}]$

  Set $IND_1 =$ index of the maximum entry of $|\boldsymbol{u}_1|$
  Initiate matrices: $\boldsymbol{W} = [\boldsymbol{u}_1]$, $\boldsymbol{P} = [\boldsymbol{e}_{IND_1}]$, $\boldsymbol{IND} = [IND_1]$
  for $i = 2, \dots, m$ do
      Solve $\boldsymbol{P}^{\mathrm{T}}\boldsymbol{W}\,\boldsymbol{c} = \boldsymbol{P}^{\mathrm{T}}\boldsymbol{u}_i$ for $\boldsymbol{c}$
      Compute residual $\boldsymbol{r} = \boldsymbol{u}_i - \boldsymbol{W}\,\boldsymbol{c}$
      Set $IND_i =$ index of the maximum component of $|\boldsymbol{r}|$
      Augment $\boldsymbol{W} \leftarrow [\boldsymbol{W}\,,\ \boldsymbol{u}_i]$ , $\boldsymbol{P} \leftarrow [\boldsymbol{P}\,,\ \boldsymbol{e}_{IND_i}]$ and $\boldsymbol{IND} \leftarrow [\boldsymbol{IND}\,,\ IND_i]$
  end
```

In Algorithm 3, $\boldsymbol{e}_i$ is the ith column of an identity matrix. The notation $\boldsymbol{M} \leftarrow [\boldsymbol{M}\,,\ \boldsymbol{v}_i]$ indicates that the matrix $\boldsymbol{M}$ is augmented by adding the column vector $\boldsymbol{v}_i$.

2.3.3. Localized Discrete Empirical Interpolation Method

DEIM succeeds in defining an efficient reduced order model, as in [6], only if the singular values of the force vector decay sufficiently fast –this effectively means that, when a relatively small number of basis vectors are considered, the sum of the squares of the singular values corresponding to the neglected basis vectors becomes negligibly small. However, in the context of a viscoplasticity constitutive model this assumption is, apparently, not valid. As it will be shown in Section 2.4.2, a considerably large number of basis vectors are required to sufficiently reduce the projection error. Hence, an efficient reduced order model cannot be constructed merely by applying the standard version of DEIM.

We employ a Localized DEIM [23] to remedy the above shortcoming. The general idea is to group samples, which are close to each other in a certain sense, into clusters. We show that the decay of the singular values becomes significantly faster for each cluster, implying that the samples are attracted to a considerably lower-dimensional subspace. In this work we only investigate the performance of DEIM in conjunction with the clustered force samples. Nonetheless, the displacement samples could also be clustered as suggested in [28]. For the purpose of clustering the samples, we make use of machine learning techniques such as the k-means clustering algorithm (for clustering samples in the offline phase), and the nearest neighbor classifier (for finding the correct cluster in the online phase).

Modified k-means clustering algorithm

We now describe a procedure to divide a set of samples $\boldsymbol{F}^{\text{samp}}$, schematically shown in Figure 2.1a, into n_c clusters with the intention that samples in each cluster are close to each other according to a certain distance definition. For this purpose, a k-means clustering algorithm, informally explained in [29], is used. In this contribution the k-means clustering algorithm is tailored for DEIM and the path-dependent nature of the problem. More specifically, we will improve on the cluster initiation, the distance definition, and the distance metric.

The strong path-dependency of the problem under consideration leads to a rather straightforward choice of the initial guess for the clustering. In other words, since force vectors from a certain part of the loading history are already "close" to each other, the samples are simply divided into n_c clusters without any permutation as an initial guess for the k-means clustering algorithm.

In contrast to the definition of the distance in the standard k-means clustering algorithm, which is the distance between a sample and the center of clusters, the distance is here defined between a sample and its DEIM approximation. For instance, consider a sample $\boldsymbol{f}_i$ and, for each cluster $c = 1 \ldots n_c$, a set of basis vectors ${}_c\boldsymbol{U}$ and a matrix ${}_c\boldsymbol{P}$ (from here onward, the number of clusters is indicated by a subscript on the left-hand side of the corresponding quantity). The distance measure used in the k-means clustering algorithm is defined by the difference between $\boldsymbol{f}_i$ and its DEIM approximation ${}_c\hat{\boldsymbol{f}}_i = {}_c\boldsymbol{A}\,{}_c\boldsymbol{P}^{\mathrm{T}}\,\boldsymbol{f}_i$ for each cluster $c = 1 \ldots n_c$, where ${}_c\boldsymbol{A} = {}_c\boldsymbol{U}\,({}_c\boldsymbol{P}^{\mathrm{T}}{}_c\boldsymbol{U})^{-1}$. In the standard k-means clustering algorithm, it is typical to consider a Euclidean distance metric. Shortcomings of the Euclidean distance are further discussed in [30]. Here, a stricter metric, based on the Euclidean distance and the cosine similarity, is defined:

$$d = \sqrt{d_{\text{Euclidean}}^2 + d_{\text{cosine}}^2}, \tag{2.28}$$

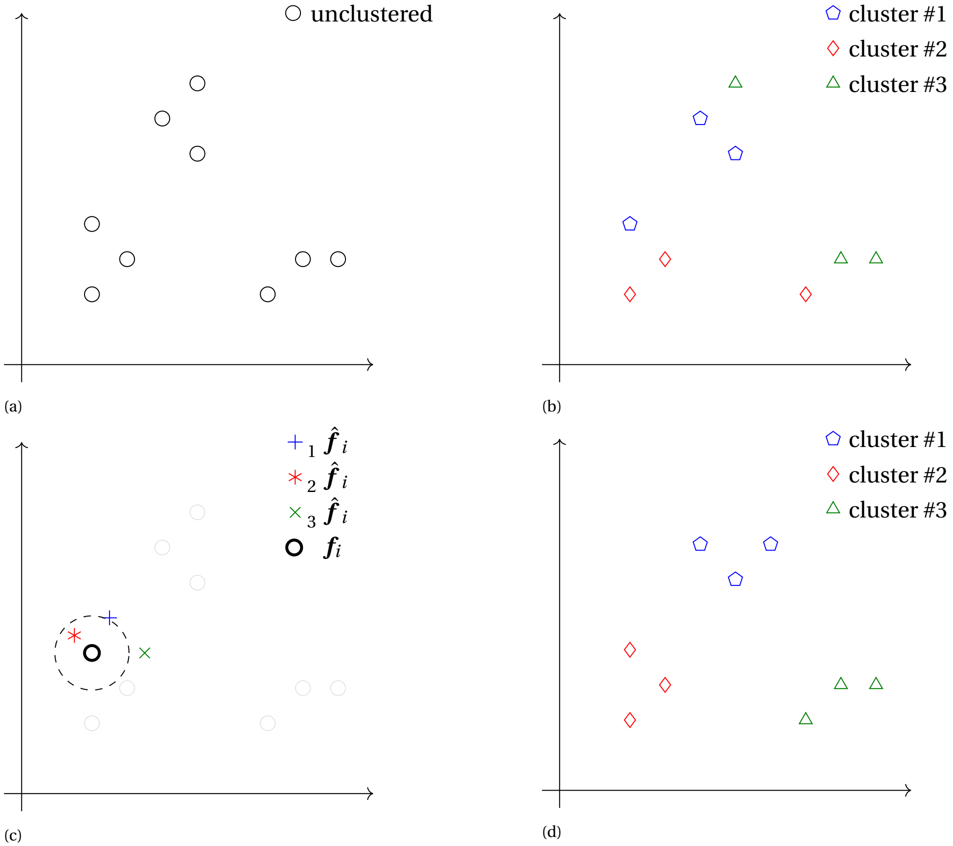
where

$$d_{\text{Euclidean}} = \|\boldsymbol{f}_i -{}_c\hat{\boldsymbol{f}}_i\|_2 \,/\, \|\boldsymbol{f}_i\|_2 \quad \text{and} \quad d_{\text{cosine}} = 1 - |\boldsymbol{f}_i^T\,{}_c\hat{\boldsymbol{f}}_i| \,/\, (\|\boldsymbol{f}_i\|_2\ \|{}_c\hat{\boldsymbol{f}}_i\|_2). \tag{2.29}$$

Since $\|\boldsymbol{f}_i -{}_c\hat{\boldsymbol{f}}_i\|_2 < \|\boldsymbol{f}_i\|_2$, these two measures are comparable in terms of magnitude; values of $d_{\text{Euclidean}}$ and d_{cosine} fall between zeros and one, and it is therefore safe to take their norm as in (2.28).

The proposed k-means clustering algorithm is detailed in Algorithm 4. The procedure initiates by dividing samples in $\boldsymbol{F}^{\text{samp}}$ into n_c clusters, as schematically shown in Figure 2.1b, without permuting them. Additionally, a mapping ($\boldsymbol{sampleToClus}$: sample → cluster number) is initiated to keep track of samples and their corresponding cluster number. In the k-means iterative scheme, the POD (Algorithm 1) and the DEIM (Algorithm 3) procedures are applied to each one of the clusters, and a reduced set of basis vectors ${}_c\boldsymbol{U}$, a matrix ${}_c\boldsymbol{P}$, and a set of DEIM selected entries ${}_c\boldsymbol{IND}$ are computed. The distance between each sample $\boldsymbol{f}_i$ and its DEIM approximation ${}_c\hat{\boldsymbol{f}}_i = {}_c\boldsymbol{A}\,{}_c\boldsymbol{P}^{\mathrm{T}}\,\boldsymbol{f}_i$ (2.28) is saved in a matrix $\boldsymbol{G} \in \mathbb{N}^{n_s \times n_c}$ (n_s being the number of samples).

Next, each sample $\boldsymbol{f}_i$ is checked to make sure that it belongs to its cluster. To do so, ${}_c\,\hat{\boldsymbol{f}}_i = {}_c\boldsymbol{A}\;{}_c\boldsymbol{P}^{\mathrm{T}}\;\boldsymbol{f}_i$ is computed for each one of the clusters $c = 1\;\ldots\;n_c$. Then, the distance between $\boldsymbol{f}_i$ and each one of the ${}_c\,\hat{\boldsymbol{f}}_i$ is measured by means of (2.28). The cluster that helps generating the most accurate DEIM approximation of $\boldsymbol{f}_i$, such that ${}_c\,\hat{\boldsymbol{f}}_i$ is closest to $\boldsymbol{f}_i$ as shown in Figure 2.1c, is the cluster $\boldsymbol{f}_i$ actually belongs to. To keep track of samples in the clusters, when a sample is designated to another cluster, the mapping ***sampleToClus*** is updated. The procedure continues until a maximum number of iterations is reached or all the samples are (sub)optimally clustered.


○ unclustered
⬠ cluster #1
◇ cluster #2
△ cluster #3
$+\;{}_1\,\hat{\boldsymbol{f}}_i$
$*\;{}_2\,\hat{\boldsymbol{f}}_i$
$\times\;{}_3\,\hat{\boldsymbol{f}}_i$
$\circ\;\boldsymbol{f}_i$
⬠ cluster #1
◇ cluster #2
△ cluster #3

(a)

(b)

(c)

(d)

Figure 2.1: Modified k-means clustering algorithm: (a) unclustered samples; (b) initial clustering of samples; (c) distance between a sample $\boldsymbol{f}_i$ and its DEIM approximation using ${}_c\boldsymbol{A}$ and ${}_c\boldsymbol{P}$ of all clusters, $c = 1\;\ldots\;n_c$. The sample $\boldsymbol{f}_i$ belongs to the cluster that generates the closest ${}_c\,\hat{\boldsymbol{f}}_i = {}_c\boldsymbol{A}\;{}_c\boldsymbol{P}^{\mathrm{T}}\;\boldsymbol{f}_i$ in the sense of (2.28); (d) final form of clusters after convergence of the modified k-means clustering algorithm.

When an insufficient number of basis vectors is considered, the rate of convergence of the k-means clustering algorithm becomes very slow and the quality of the clusters diminishes. From the implementation point of view, to prevent division by zero in d_{cosine} and $d_{\text{Euclidean}}$ (2.29), it is vital to eliminate zero vectors from the matrix of force samples.

In order to check the validity of the proposed approach we can plot the singular values associated with each cluster and observe their rate of decay. If their rate of decay is not substantially faster than that of the singular values associated with all the samples, the number of clusters needs to be increased at the cost of a more expensive k-means clustering algorithm.

Algorithm 4: Modified k-means clustering algorithm

```
input  : A set of force vector samples $\boldsymbol{F}^{\text{samp}} = [\boldsymbol{f}_1, \ldots, \boldsymbol{f}_{n_s}]$,
         total number of clusters $n_c$,
         maximum number of iterations $maxIter$
output: $\boldsymbol{sampleToClus} : i \rightarrow c, \quad i \in 1,...,n_s, \quad c \in 1,...,n_c$

  Cluster $\boldsymbol{F}^{\text{samp}}$ without sorting into $\{{}_1\boldsymbol{F}, \ldots, {}_{n_c}\boldsymbol{F}\}$
  Initiate $\boldsymbol{sampleToClus}$
  $converged := 1$
  while $converged \neq 1$ and $maxIter$ not reached do
      for $c = 1 \ldots n_c$ do
          ${}_c\boldsymbol{U} =$ POD $({}_c\boldsymbol{F})$                                   Algorithm 1
          ${}_c\boldsymbol{IND}, {}_c\boldsymbol{P} =$ DEIM $({}_c\boldsymbol{U})$                      Algorithm 3
          ${}_c\boldsymbol{A} = {}_c\boldsymbol{U}\,({}_c\boldsymbol{P}^T\,{}_c\boldsymbol{U})^{-1}$
          for $i = 1$ to $n_s$ do
              ${}_c\hat{\boldsymbol{f}}_i = {}_c\boldsymbol{A}\,{}_c\boldsymbol{P}^{\mathrm{T}}\,\boldsymbol{f}_i$
              $\boldsymbol{G}(i,c) := d$                                       Equation (2.28)
          end
      end
      for $i = 1$ to $n_s$ do
          $c_{\text{old}} := \boldsymbol{sampleToClus}(i)$
          $c_{\text{new}} := \arg\min \boldsymbol{G}(i,:)$
          if $c_{\text{old}} = c_{\text{new}}$ then
              $converged = 1$
          else
              $converged := 0$
              $\boldsymbol{sampleToClus}(i) := c_{\text{new}}$
          end
      end
  end
```

Nearest neighbor classifier

To find the cluster that best represents the current state of the system during the runtime, we employ a nearest neighbor classifier and modify its distance metric according to (2.28). At each time step, the nearest neighbor classifier computes distances (2.28) between the converged force vector $\hat{\boldsymbol{f}}$ and each one of the clustered samples $\boldsymbol{f}_i$, $i = 1 \ldots n_s$.

The classifier then determines the sample with the least distance to the converged force vector. The cluster that contains this sample best describes the current state of the system.

To reduce the computational cost of the classification, the nearest neighbor classifier uses a low-dimensional representation of the force vector. In this context, we employ the DEIM-based feature extraction [23]

$$\hat{\boldsymbol{f}}_{\text{FE}} = {}_{c_{\text{old}}}\boldsymbol{P}^T\,\hat{\boldsymbol{f}} \quad \text{and} \quad \boldsymbol{f}_{\text{FE},i} = {}_{c_{\text{old}}}\boldsymbol{P}^T\,\boldsymbol{f}_i, \tag{2.30}$$

where $\hat{\boldsymbol{f}}_{\text{FE}} \in \mathbb{R}^m$ and $\boldsymbol{f}_{\text{FE},i} \in \mathbb{R}^m$ are a low-dimensional representation of $\hat{\boldsymbol{f}}$ and $\boldsymbol{f}_i$, respectively, and ${}_{c_{\text{old}}}\boldsymbol{P} \in \mathbb{R}^{N\times m}$ is the DEIM matrix from the previous time step. Distance metric in (2.28) are computed as

$$d_{\text{Euclidean}} = \|\boldsymbol{f}_{\text{FE},i} - \hat{\boldsymbol{f}}_{\text{FE}}\|_2 \,/\, \|\boldsymbol{f}_{\text{FE},i}\|_2 \quad \text{and} \quad d_{\text{cosine}} = 1 - |\boldsymbol{f}^{\text{T}}_{\text{FE},i}\;\hat{\boldsymbol{f}}_{\text{FE}}| \,/\, (\|\boldsymbol{f}_{\text{FE},i}\|_2\;\|\hat{\boldsymbol{f}}_{\text{FE}}\|_2). \tag{2.31}$$

.

Algorithm 5: Nearest neighbor classifier

```
input  : A set of force vector samples $\boldsymbol{F}^{\text{samp}} = [\boldsymbol{f}_1, \dots, \boldsymbol{f}_{n_s}]$,
         $\boldsymbol{sampleToClus}: i \to c, \quad i \in 1, ..., n_s, \quad c \in 1, ..., n_c$,
         the force vector $\hat{\boldsymbol{f}} \in \mathbb{R}^N$ at time step $t$,
         the current cluster $c_{\text{old}}$
output: New cluster $c_{\text{new}}$

    $\hat{\boldsymbol{f}}_{\text{FE}} := {}_{c_{\text{old}}}\boldsymbol{P}^T\,\hat{\boldsymbol{f}}$
    for $i = 1 \dots n_s$ do
        $\boldsymbol{f}_{\text{FE},i} := {}_{c_{\text{old}}}\boldsymbol{P}^T\,\boldsymbol{f}_i$
        $\boldsymbol{d}(i) := d(\hat{\boldsymbol{f}}_{\text{FE}}, \boldsymbol{f}_{\text{FE},i})$                Equations (2.28) and (2.31)
    end
    $i_{\text{min}} := \text{arg min}(\boldsymbol{d})$
    $c_{\text{new}} := \boldsymbol{sampleToClus}(i_{\text{min}})$
```

2.3.4. DISCUSSION ON THE COMPUTATIONAL COST

The computational cost of DEIM and LDEIM is already discussed in [6, 23]. For the sake of completeness, this section is dedicated to a brief review of the key points. Moreover, we also discuss some of our observations.

- In spite of the additional computational cost of the clustering, the cost of the offline phase is still strongly dominated by the sample collection.
- For the online phase, the primary contributor to the computational cost is the nearest neighbor classifier, Algorithm 5. Note that the computational complexity of this procedure is independent of the number of dofs N, thanks to the feature extraction concept (2.30). However, it strongly depends on the total number of

samples n_s. As n_s becomes larger, the nearest neighbor classification loses its efficiency. To tackle this problem, it is possible to either reduce the number of samples through Redundancy Reduction methods such as the QR-SVD technique [31, 32], or perform the classification based on a set of surrogate quantities instead of samples. For instance, the centroid of the clusters computed in classical k-means clustering algorithm can be an adequate substitute. In this manner, the procedure becomes dependent on the number of clusters. However, hopefully, the number of clusters are always significantly smaller than the number of samples.

- If, as a result of the POD and the DEIM approximations, the consistency of the linearization of (2.17) is jeopardized, the number of iterations in the Newton-Raphson scheme in the reduced order model will increase. Nevertheless, the iteration counts could be kept the same by accepting a solution that is not fully converged. This claim could be conjectured as follows. Consider the error between the full order model and the outcome of the Newton-Raphson scheme in the reduced order model, $\boldsymbol{e} = \boldsymbol{a} - \boldsymbol{a}^{\mathrm{NR}}$. By adding and subtracting the exact solution of (2.17), $\boldsymbol{a}^{\mathrm{ROM}}$, to the right hand side of the error expression and taking the L_2 norm of both sides, an upper bound to the error becomes

  $$\|\boldsymbol{e}\|_2 \leq \|\boldsymbol{a} - \boldsymbol{a}^{\mathrm{ROM}}\|_2 + \|\boldsymbol{a}^{\mathrm{ROM}} - \boldsymbol{a}^{\mathrm{NR}}\|_2. \tag{2.32}$$

  From the equation above, it can be inferred that, regardless of how accurately (2.17) is solved, even at the limit where $\|\boldsymbol{a}^{\mathrm{ROM}} - \boldsymbol{a}^{\mathrm{NR}}\|_2$ becomes effectively zero, the error as a result of the MOR approximation $\|\boldsymbol{a} - \boldsymbol{a}^{\mathrm{ROM}}\|_2$ remains untouched. As a result, the bound on the error is, at most, as tight as $\|\boldsymbol{a} - \boldsymbol{a}^{\mathrm{ROM}}\|_2$ permits. Hence, a solution that is relatively accurate, considering the error as a result of MOR, but not fully converged in the Newton-Raphson scheme should be considered as acceptable. Nevertheless, in Section 3.8, we always consider enough basis vectors to ensure the consistency of the linearization and, consequently, quadratic convergence rate of the Newton-Raphson scheme.

- In theory, as the number of clusters becomes larger, the computer memory occupied by the basis vectors in the online phase grows dramatically. However, in practice, we experienced that by increasing the number of clusters, the number of basis vectors in each cluster becomes significantly smaller. As a result, the cumulative number of basis vectors does not increase dramatically, as shown in Section 2.4.3.

2.3.5. Summary of the proposed Model Order Reduction technique

Next, a step-by-step guide for the proposed MOR technique is outlined. The offline phase (Algorithm 6) is run first and its output data are saved to a file. In the online phase (Algorithm 7), the data are read and the reduced order model is employed to run the simulation.

2.4. Applications

We demonstrate the proposed MOR technique by means of three typical solid mechanics problems. A specific set of parameters ($\eta = 10^{-5}\ \mathrm{s}^{-1}$, $a = -1$, $b = 50$, $Y = 1\ \mathrm{N}\ /\ \mathrm{mm}^{-2}$, $E = 1000\ \mathrm{N}\ /\ \mathrm{mm}^{-2}$, $\nu = 0$) is considered unless stated otherwise.

Algorithm 6: Offline phase

```
1. Collect samples $\boldsymbol{A}^{\text{samp}}$ and $\boldsymbol{F}^{\text{samp}}$using a FEM analysis.
2. $\boldsymbol{V} := \text{POD}(\boldsymbol{A}^{\text{samp}})$.                Section 2.3.1
3. $\boldsymbol{sampleToClus} := k\text{-means}(\boldsymbol{F}^{\text{samp}})$        Section 2.3.3
for $c = 1 \dots n_c$ :
    3.1. let $\boldsymbol{i}_c$ be such that $\boldsymbol{sampleToClus}(\boldsymbol{i}_c) = c$
    3.1. ${}_c\boldsymbol{F} := \boldsymbol{F}^{\text{samp}}(\boldsymbol{i}_c)$
    3.1. ${}_c\boldsymbol{U} :=$ POD $({}_c\boldsymbol{F})$,                Section 2.3.1
    3.2. ${}_c\boldsymbol{IND},\ {}_c\boldsymbol{P} := \text{DEIM}({}_c\boldsymbol{U})$,        Section 2.3.2
    3.3. ${}_c\boldsymbol{A} := {}_c\boldsymbol{U}\ ({}_c\boldsymbol{P}^T\ {}_c\boldsymbol{U})^{-1}$, and
    3.4. Save ${}_c\boldsymbol{A}, {}_c\boldsymbol{P}, {}_c\boldsymbol{IND}$.
4. Save $\boldsymbol{V}, \boldsymbol{F}^{\text{samp}}, \boldsymbol{sampleToClus}$.
```

2.4.1. TENSILE BAR: ONE-DIMENSIONAL STRAIN-LOCALIZATION

The tapered bar in Figure 2.2 is clamped at the left-hand side. The bar is subjected to a monotonic tensile loading. The final displacement of 1 mm is applied at the free end in 200 steps with a constant rate of 0.66×10^{-4} mm / s. Figure 2.3 shows the results of a mesh refinement study performed with uniform subdivisions of the bar domain along the axis. The mesh refinement study identifies the solution related to the 71 eight-node regular hexahedral element mesh as the reference solution for the development of a reduced order model.

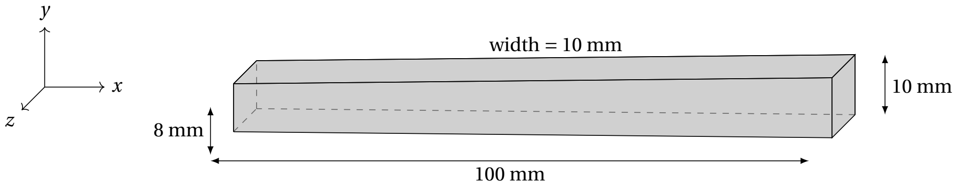


Figure 2.2: The clamped end of the tapered bar has a smaller cross-section to trigger the initiation of plasticity and consequent strain localization.

Algorithm 7: Online phase

```
1. Initiate FEM code and load:
  1.1. displacement basis vectors $\boldsymbol{V}$,                                Algorithm 6
  1.2. the sample matrix $\boldsymbol{F}^{\text{samp}}$,                              Algorithm 6
  1.3. the mapping $\boldsymbol{sampleToClus}$,
  1.4. ${}_{c}\boldsymbol{A}, {}_{c}\boldsymbol{P}, {}_{c}\boldsymbol{IND}$ for all clusters $(c = 1 \dots n_c)$.        Algorithm 6
2. Set the cluster $c_{\text{new}} = 1$.
3. For each time step $t$ :
  3.1. For each Newton-Raphson iteration $j$ :
    3.1.0. compute the linear part of the stiffness matrix $\boldsymbol{K}^{\text{L}}$,   Equation (2.14)
    3.1.1. $\hat{\boldsymbol{f}} := {}_{c_{\text{new}}}\boldsymbol{A}\ \boldsymbol{f}_{\text{DEIM}}({}_{c_{\text{new}}}\boldsymbol{IND})$,          Section 2.3.2
    3.1.2. $\hat{\boldsymbol{K}} := {}_{c_{\text{new}}}\boldsymbol{A}\ \boldsymbol{K}_{\text{DEIM}}({}_{c_{\text{new}}}\boldsymbol{IND})$,          Section 2.3.2
    3.1.3. $\tilde{\boldsymbol{r}}^{j} := \boldsymbol{V}^{\text{T}}\ (\boldsymbol{K}^{\text{L}}\ \boldsymbol{a}^{j} + \hat{\boldsymbol{f}})$,                Section 2.3.1
    3.1.4. $\tilde{\boldsymbol{K}}^{j} := \boldsymbol{V}^{\text{T}}\ (\boldsymbol{K}^{\text{L}} + \hat{\boldsymbol{K}})\ \boldsymbol{V}$,                Section 2.3.1
    3.1.5. solve $\tilde{\boldsymbol{K}}^{j}\ \mathrm{d}\tilde{\boldsymbol{a}}^{j+1} = -\tilde{\boldsymbol{r}}^{j}$,                  Equation (2.18.1)
    3.1.6. $\tilde{\boldsymbol{a}}^{j} \leftarrow \tilde{\boldsymbol{a}}^{j} + \mathrm{d}\tilde{\boldsymbol{a}}^{j+1}$,                         Equation (2.18.2)
    3.1.7. check convergence: $\tilde{\boldsymbol{r}}^{j} < tol$.
  3.2. Set the cluster $c_{\text{old}} = c_{\text{new}}$.
  3.3. get $c_{\text{new}}$using the nearest neighbor classifier              Section 2.3.3
  3.4. Go to the next time step.
```

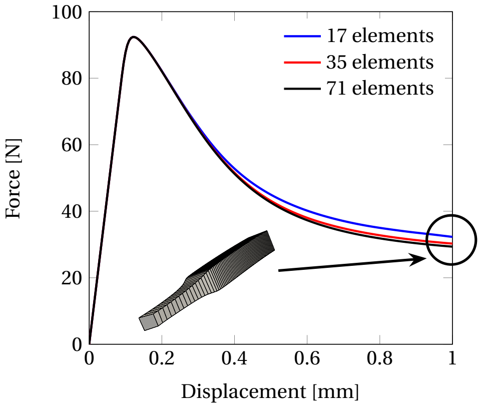


Figure 2.3: The regularizing properties of viscoplasticity are reflected in the mesh refinement study in terms of the resultant force at the clamped end plotted against the displacement at the free end.

To test the MOR technique, we consider three different values of the b parameter ($b = [0, 20, 100]$). According to (2.8), large positive values of b result in a fast decay of σ_Y as κ grows and characterize the softening branch of the load-displacement curve as shown in Figure 2.3. A reference solution is computed for each value of b, and the corresponding displacement and force samples are collected in matrices $\boldsymbol{A}^{\text{samp}} = [\boldsymbol{a}_1 \; \dots \; \boldsymbol{a}_{200}]$ and $\boldsymbol{F}^{\text{samp}} = [\boldsymbol{f}_1 \; \dots \; \boldsymbol{f}_{200}]$.

A reduced order model is constructed using these samples for each value of the b parameter. The result of the online computation, depicted in Figure 2.4, evidently show the complexity reduction achieved by employing our MOR technique. For instance, for the steepest strain softening curve, obtained with $b = 100$, the size of the system of equations is reduced, without apparent loss of accuracy, to 93 × 93 from 864 × 864 and, in the light of DEIM, evaluation of the force vector is required to be carried out at 57 entries rather than 864. A subdivision of the samples into three clusters allows to further reduce this number to 63-17-15 (the notation 63-17-15 indicates that we have considered three clusters and that the first cluster requires the evaluation of 63 entries, the second requires 17 entries, and the last only 15). For the other two curves, the size of the system of equations is reduced to 115 × 115 ($b = 20$) and 94 × 94 ($b = 0$). The evaluation of the force vector is carried out at 30 and 53 entries, respectively, when employing DEIM. With LDEIM and three clusters, it is possible to further reduce the number of entries to 32-11-8 and 51-22-13, respectively.

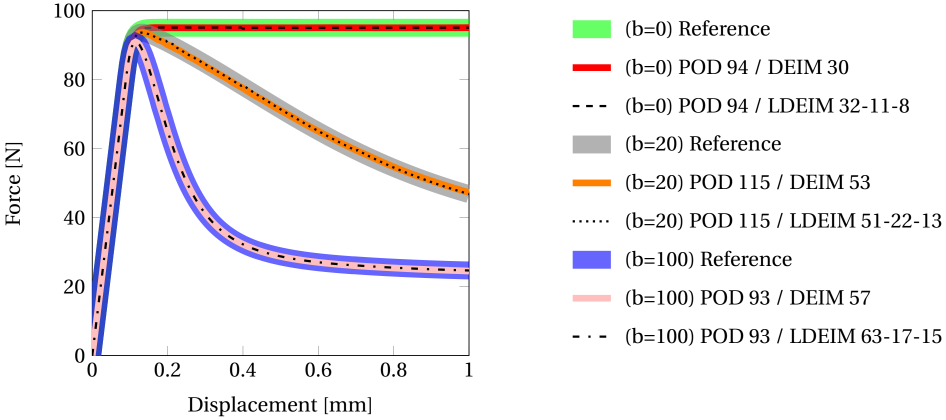


Figure 2.4: The MOR technique applied to the one-dimensional strain-localization problem. The reference curves are obtained with the 71 eight-node regular hexahedral element mesh (864 DOFs) using different values of the b parameter (2.8).

2.4.2. WALL UNDER COMPRESSION: SHEAR BAND FORMATION

This example is dedicated to a wall under compression that shows a localization zone in form of a shear band. The lower and the left-hand side edges of the wall depicted in Figure 2.5 are constrained in the x and y directions. The wall is subjected to a monotonic compressive loading. The final displacement of 2 mm is applied at the upper edge in 400 steps with a constant rate of 1.33×10^{-4} mm / s. The parameter b in (2.8) is taken equal to 100.

A mesh refinement study has been carried out considering uniform grids with one element across the thickness. Figure 2.6 shows the results of the mesh refinement study. The solution obtained with 196 (14×14) eight-node regular hexahedral elements is taken as the reference solution.

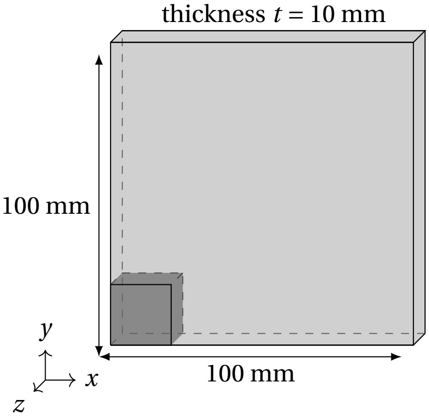


Figure 2.5: Geometry of the wall under compression with the dark shaded part indicating the weak region that localizes the initiation of plasticity and triggers a shear band.

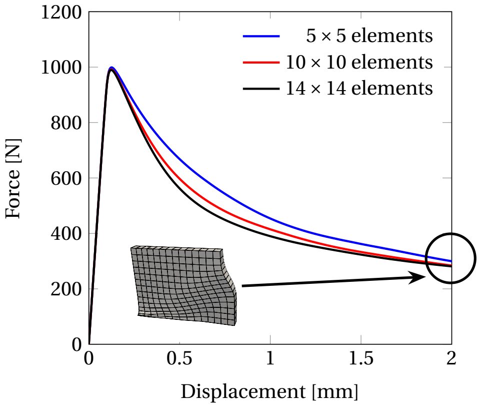


Figure 2.6: The regularizing properties of viscoplasticity are reflected in the mesh refinement study in terms of the resultant of the force in the vertical direction on the lower edge against the value of the displacement of the upper edge; the inset shows the deformed wall with the shear band due to strain localization.

Displacement and force samples are collected at each time step in matrices $\boldsymbol{A}^{\text{samp}} = [\boldsymbol{a}_1 \ \dots \ \boldsymbol{a}_{400}]$ and $\boldsymbol{F}^{\text{samp}} = [\boldsymbol{f}_1 \ \dots \ \boldsymbol{f}_{400}]$. Figure 2.7a shows the convergence of DEIM to the reference solution by a basis addition study. The number of the basis vectors is chosen based on the sum of the square of the neglected singular values as explained in Section 2.3.1. It is possible to make an analogy between adding the basis vectors associated with the singular values shown in Figure 2.7b for a basis addition study and increasing the number of elements for a mesh refinement study. By adding each subsequent basis vector, the squared value of its associated singular value (say σ_m) is deducted from the error defined in (2.21) –the error then becomes $\sum_{i=m+1}^{r} \sigma_i^2$ in which the contribution of σ_m^2 is missing thus yielding a smaller value.

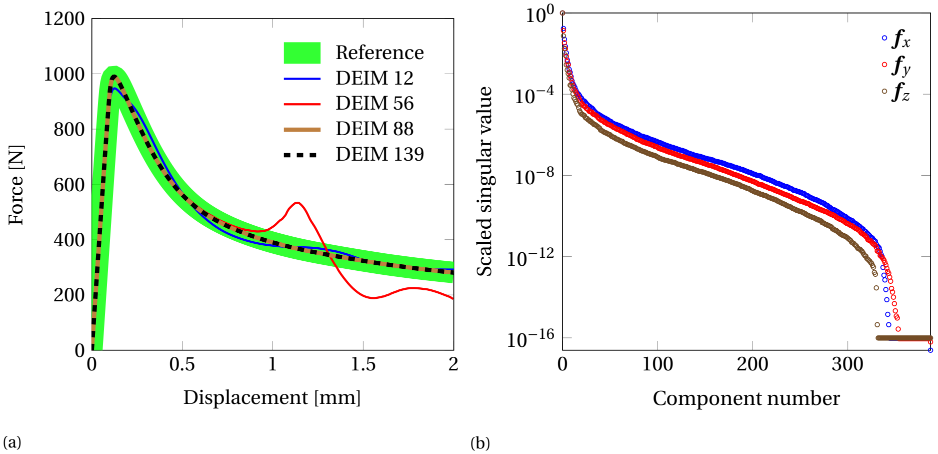


Figure 2.7: Performance of the DEIM approximation in the two-dimensional strain localization problem: (a) basis addition study (reference solution: 1350 DOFs, 196 elements; reduced order model: 227 POD basis vectors associated with the displacement field); (b) singular values of the force vector samples.

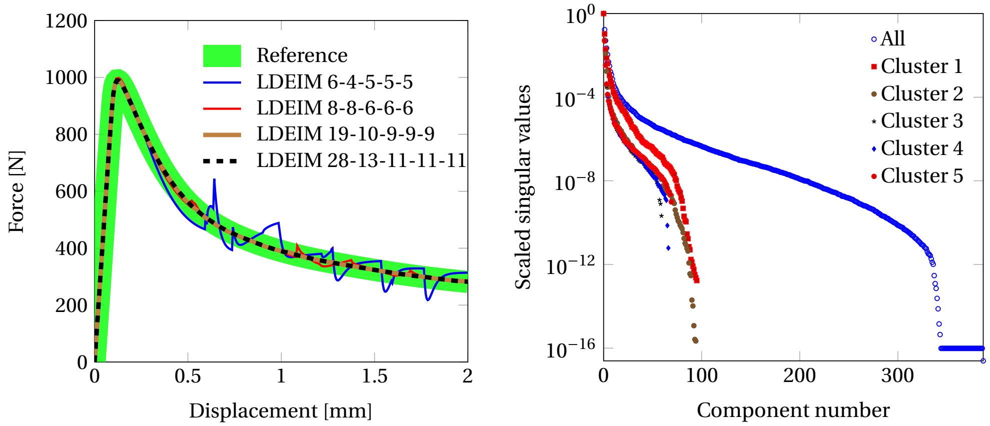
Figure 2.7b evidently shows the slow rate of decay of the singular values. As a result, a large number of basis vectors is required to decrease the projection error, that is the sum of the square of the neglected singular values, to an acceptable value and render the reduced order model sufficiently accurate. Figure 2.7a shows that at least 88 DEIM entries (brown curve) are required to construct a reduced order model that is capable of reproducing the reference solution (green curve). The black dashed line is to confirm that we have reached the converged solution. Checking the difference between results of two consecutive basis addition cases can be an adequate criterion to determine the convergence of a reduced order model. It is also noteworthy to report a case where the DEIM approximation results in an unstable reduced order model as shown by the red curve in Figure 2.7a. The source of this instability is, to the best of our knowledge, still unknown. The same type of behavior is reported in [32] and [33].

The force vector and the stiffness matrix associated with these 88 DOFs are computed by evaluating all the elements shown in Figure 2.9a, clearly a major subset of the elements in the mesh. As a result, significant reduction in computation cost can not be achieved. To mitigate this issue, we cluster the samples of the force vector that are 'close' to each other and compute, for each cluster, a set of reduced basis vectors and a set of DEIM selected entries as discussed in Section 2.3.3. As shown in Figure 2.8a, a basis addition study is carried out with 5 clusters. The reduced order model with 19-10-9-9-9 DEIM entries (brown curve) accurately approximates the reference solution (green curve). The singular values corresponding to the x component of the force vector samples are depicted in Figure 2.8b for each cluster of the reduced order model (those corresponding to the y and the z components follow the same trend). It is clear that the singular values associated with each cluster decay significantly faster than those of all samples.

The elements used in this case are depicted in Figure 2.9b. Due to the path-dependent nature of the model, all elements depicted in this figure need to be updated for their history parameters. However, only a subset of them is used to compute the force vector and the stiffness matrix at each Newton-Raphson iteration. Figures 2.9a and 2.9b show that the online phase of LDEIM is computationally cheaper than that of DEIM.

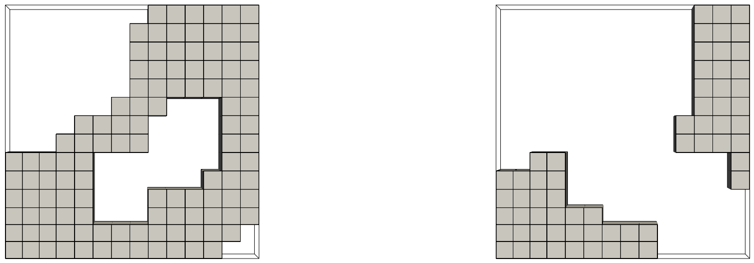
As discussed in Section 2.3.3, a nearest neighbor classifier is executed at each time step to find the cluster the current force vector belongs to. The classification is carried out based on the $m \ll N$ DEIM-selected entries of the force vector. Due to the non-periodic nature of the mechanical response in this class of problems, clusters do not get repeated during the run time. For instance, if the first cluster is selected at a time step close to the peak of the load-displacement curve, it will not be selected when the load-displacement curve reaches a plateau at the end. Cluster selection being automatic, this observation can be used to monitor the consistency of cluster switching. In other words, repetition of a cluster during the simulation is an indication that either the clustering procedure in the offline phase or the classification procedure in the online phase is inconsistent with the nature of the path-dependent problem. Additionally, this observation can be used to increase the efficiency of the nearest neighbor classifier, Algorithm 5. Indeed, by neglecting samples associated with previously selected clusters, the classifier can compare the current state of the system to fewer number of samples.

To achieve consistency as explained in the previous paragraph, we modify the distance metric in the k-means clustering algorithm and the nearest neighbor classifier



(a) (b)

Figure 2.8: Performance of the LDEIM approximation in the two-dimensional strain localization problem: (a) basis addition study (reference solution: 1350 DOFs, 196 elements; reduced order model: 227 POD basis vectors associated with the displacement field); (b) singular values of the x component of the force vector samples using unclustered and clustered samples (those corresponding to the y and the z components follow the same trend).



(a) (b)

Figure 2.9: Elements used in the online phase in the two-dimensional strain localization problem: (a) DEIM –brown line in Figure 2.7a; (b) LDEIM –brown line in Figure 2.8a.

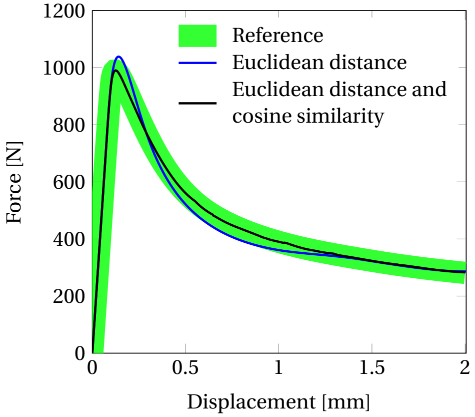


Figure 2.10: The effect of the distance metric on the load-displacement curves of the two-dimensional strain localization problem (reference solution: 1350 DOFs, 196 element; reduced order model: 227 POD basis vectors associated with the displacement field and LDEIM with 5 clusters and 19-10-9-9-9 DEIM entries in each cluster).

algorithms. Figure 2.10 shows that the reduced order model based on the Euclidean distance and the cosine similarity metric (2.28) (black curve) properly reproduces the reference solution, whereas the reduced order model based on the standard Euclidean distance definition alone fails to do so. For the Euclidean distance case, the cluster nunmber switches from 1 to 5 right before the peak load (note that cluster 5 is associated with the plateau part of the load-displacement curve). This is not consistent and, as a result, the blue curve deviates from the reference equilibrium path.

Parameter sensitivity analysis

We now demonstrate an application of the reduced order model to a simple parameter sensitivity analysis study. The goal is to construct a reduced order model response related to a set of parameters using samples obtained from two different parameter sets. To this end, we run the standard FEM model for $b = 80$ and $b = 120$ (the other parameters are defined at the beginning of Section 3.8). Samples are uniformly collected in matrices $\boldsymbol{A}_{b80}^{\text{samp}}$, $\boldsymbol{F}_{b80}^{\text{samp}}$, $\boldsymbol{A}_{b120}^{\text{samp}}$ and $\boldsymbol{F}_{b120}^{\text{samp}}$ at each time step. These samples are combined into two matrices $\boldsymbol{A}^{\text{samp}}$ and $\boldsymbol{F}^{\text{samp}}$. Since the k-means initial clustering uniformly divides samples into clusters, it is a good practice to combine $\boldsymbol{F}_{b80}^{\text{samp}}$ and $\boldsymbol{F}_{b120}^{\text{samp}}$ in a way that samples from the peak load region are collected in the initial columns of $\boldsymbol{F}^{\text{samp}}$ and samples from plateau region are collected in the last columns as described in Algorithm 8.

A reduced order model is constructed using these samples. In Figure 2.12a, the reduced order model is employed to simulate the case with $b = 100$. To approximate the standard FEM solution, 85 DEIM entries are required when the samples are not clustered. However, when the samples are divided into 5 clusters, only 31-19-19-19-17 DEIM entries are required. It is also shown that fewer DEIM entries are needed to reach an acceptably accurate solution (black curve) when the number of clusters increases.

In Figure 2.12a we observe fluctuations in the load-displacement curves of the reduced order models, starting from a displacement equal to 0.4 mm. A possible explana-

```
Algorithm 8: Construct $\boldsymbol{F}^{\text{samp}}$
    input  : Two sets of force vector samples for $b = 80$ and $b = 120$: $\boldsymbol{F}_{b80}^{\text{samp}} \in \mathbb{R}^{N\times n_s}$
             and $\boldsymbol{F}_{b120}^{\text{samp}} \in \mathbb{R}^{N\times n_s}$
    output: A mixed sample matrix $\boldsymbol{F}^{\text{samp}} \in \mathbb{R}^{N\times 2n_s}$
    Initiate the mixed sample matrix: $\boldsymbol{F}^{\text{samp}} = [\boldsymbol{F}_{b80}(:,1)\ ,\ \boldsymbol{F}_{b120}(:,1)]$
    for $i = 2 \dots n_s$ do
        Append $i$th columns of $\boldsymbol{F}_{b80}$ and $\boldsymbol{F}_{b120}$ to $\boldsymbol{F}^{\text{samp}}$
    end
```

tion could be related to the set of samples that underlies the construction of the basis vectors for the cases with $b = 80$ and $b = 120$. According to (2.8), the nonlinear parts of the stress related to these two cases, and the corresponding force vectors, differ slightly at the beginning when the plastic strain κ is small; hence, the reduced subspaces corresponding to the two force vectors are close to one another. However, as κ grows, the nonlinear part of the stress for $b = 80$ case decays slower than that for $b = 120$ and, as a result, their corresponding force vectors tend to deviate from one another. Consequently, the corresponding reduced subspaces these two cases belong to are no longer as close as they used to be. At this point, let us consider the case $b = 100$ with a reduced order model constructed using samples collected from the cases with $b = 80$ and $b = 120$. Figure 2.12a supports the hyphothesis just advanced. At the beginning, the load-displacement curves of the reduced order models are fairly close to that of the reference curve (green curve), suggesting that the force vectors for the $b = 100$ case are adequately approximated. Later on, however, small fluctuations in the load-displacement curves of the reduced order models become visible as κ grows.

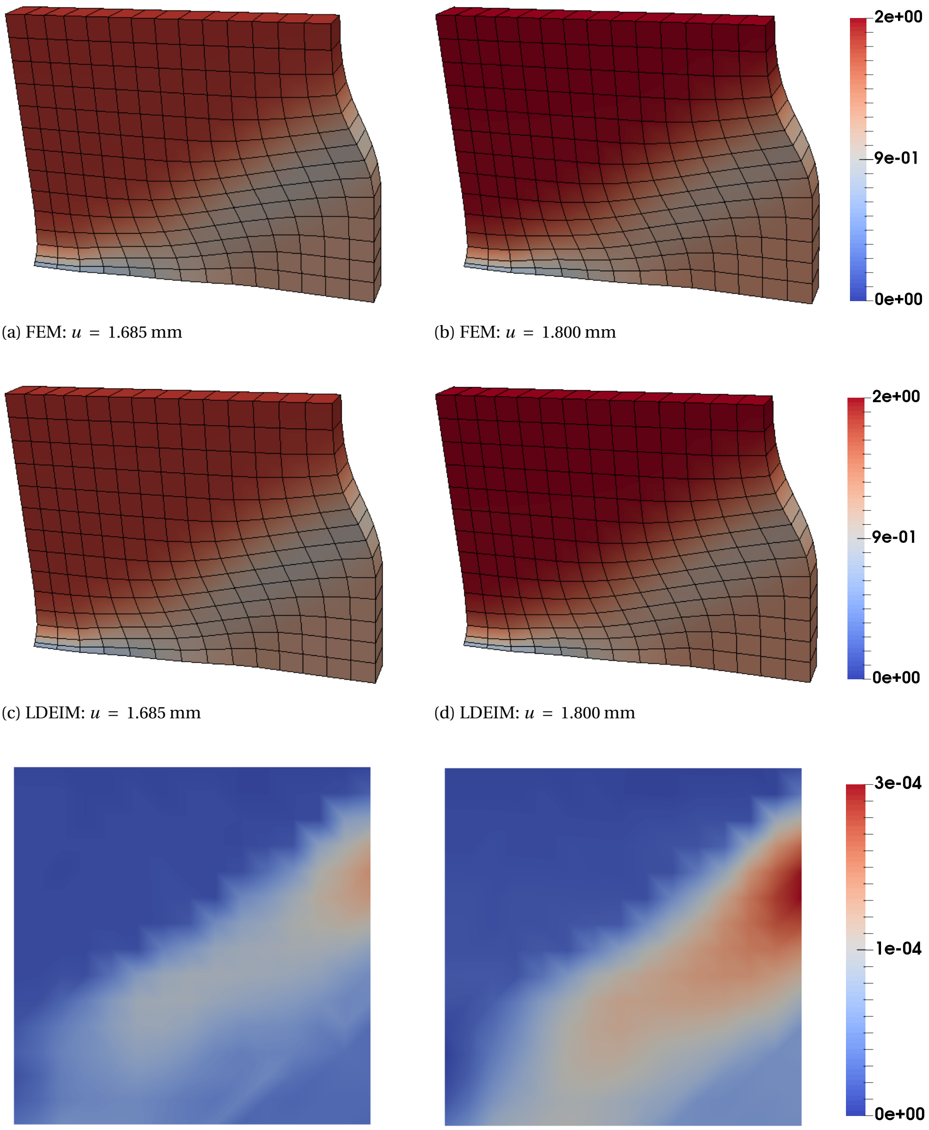
To reduce these fluctuations, either the number of the basis vectors (and consequently the DEIM entries) in each cluster should be increased or more clusters should be taken into account. The first approach decreases the efficiency of the reduced order model and is not considered. The second approach increases the accuracy of the reduced order model and permits the use of the same or even smaller number of DEIM entries as shown in Figure 2.12a.

To quantify the impact of the fluctuations in the load-displacement curve of the reduced order model with 10 clusters (black line), we consider the points that coincide with the 8th and 9th cluster change as shown in Figure 2.12b. These two points correspond to the largest deviation from the reference solution (green curve) and are used to compare their corresponding displacement fields with that of the reference solution employing the relative error measurement

$$\boldsymbol{e} = \frac{|\boldsymbol{u}_{\text{FEM}} - \boldsymbol{u}_{\text{LDEIM}}|}{\|\boldsymbol{u}_{\text{FEM}}\|_2}, \tag{2.33}$$

where $|\boldsymbol{\Theta}|$ is the component-wise absolute value of $\boldsymbol{\Theta}$. Figure 2.11 shows the deformed specimen and the relative error corresponding to these two points. It can be observed that the reduced order model is sufficiently accurate in the entire domain with maximum relative error of 3×10^{-4}.

It is noteworthy that we do not enforce clusters to change consecutively. However, we would expect them to change in this manner due to the special initial k-means clustering discussed in Section 2.3.3.



(a) FEM: $u = 1.685$ mm

(b) FEM: $u = 1.800$ mm

(c) LDEIM: $u = 1.685$ mm

(d) LDEIM: $u = 1.800$ mm

(e) Relative error at $u = 1.685$ mm

(f) Relative error at $u = 1.800$ mm

Figure 2.11: Two-dimensional strain localization problem: deformed configuration corresponding to the black curve depicted in Figure 2.12a at displacement levels corresponding to the 8th and 9th cluster changes indicated in Figure 2.12b. The reference FEM solution and reduced order model are depicted in the upper rows and the error between them is depicted in the bottom row.

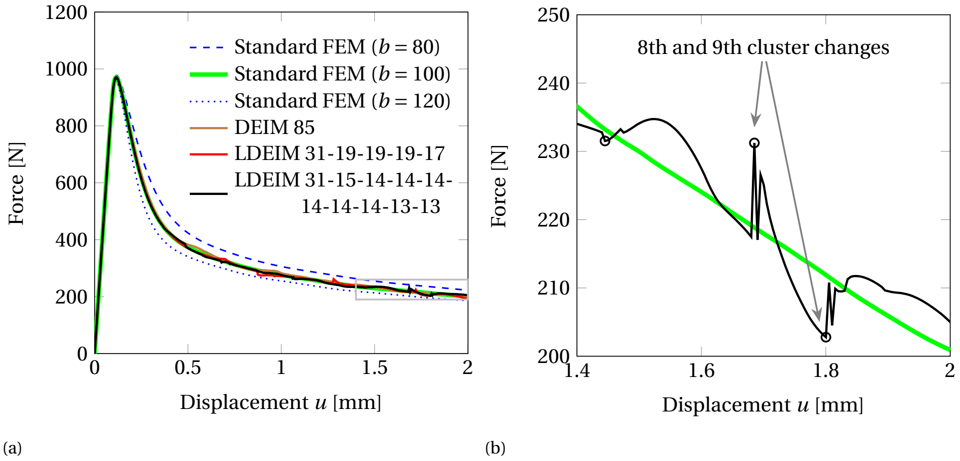


Figure 2.12: Simple parameter sensitivity analysis for the two-dimensional strain localization problem: (a) evaluation of the case with $b = 100$ using the reduced order model constructed from samples collected from reference cases with $b = 80$ and $b = 120$ (reduced order model: 325 POD basis vectors associated with the displacement field for DEIM and LDEIM); (b) enlargement of the boxed region in (a) with locations of cluster change marked.

2.4.3. THREE-DIMENSIONAL SQUARE FOOTING

The footing depicted in Figure 2.13 is subjected to a monotonic compressive loading. The final compressive displacement of 1 mm is applied at the dark shaded plate on the upper surface in 1200 steps with a constant rate of 0.66×10^{-4} mm / s. The surface on the left-hand side, bottom, and the front are constrained from moving in the x, the z and the y directions, respectively.

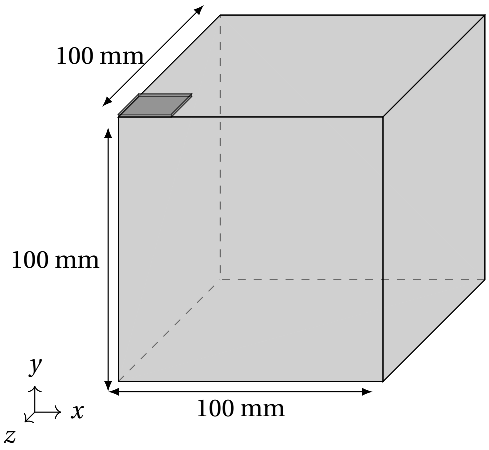


Figure 2.13: Geometry.

A mesh refinement study is carried out and the result is depicted in Figure 2.14. Load-displacement curves are associated with the resultant of the forces on the y plane at the bottom and the vertical displacement of the loading plate. Here, we choose the load-displacement curve obtained with 1331 eight-node hexahedral elements as the reference solution. The numerical solution, although not yet converged, is sufficiently accurate for the purpose of this study.

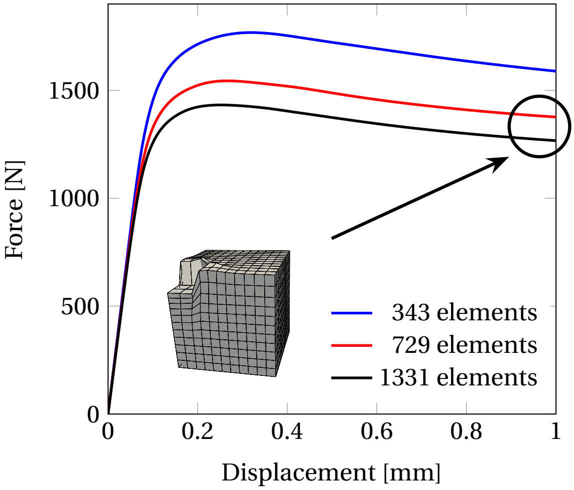


Figure 2.14: Mesh refinement study.

In total, 1200 samples were uniformly collected from the reference solution to construct a reduced order model. A large number of POD basis vectors (609), associated with the displacement field, is considered to sufficiently reduce the POD-G projection error. In this manner, it is possible to objectively study the performance of the DEIM approximation.

As shown in Figure 2.15, at least 335 DEIM entries are required to almost reconstruct the reference solution. Note that the load-displacement curve of the reduced order model starts to deviate from the reference solution before reaching the final displacement value. Nevertheless, it is possible to considerably reduce the number of the DEIM entries by clustering samples. For instance, considering 20, 30, and 40 clusters at most, as depicted in Figure 2.16, 117, 101, and 43 DEIM entries are required, respectively. This observation implies that by increasing the number of clusters it is possible to construct a significantly more efficient reduced order model.

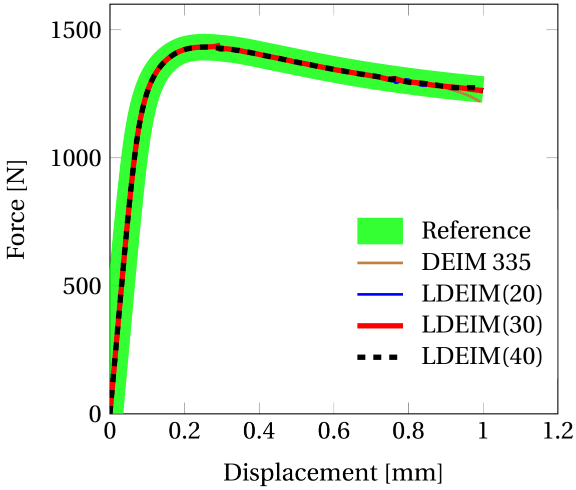


Figure 2.15: Footing: load-displacement curves associated with different number of clusters (reference solution: 5184 DOFs, 1331 elements; reduced order model: 609 POD basis vectors associated with the displacement field for DEIM and LDEIM).

In practice, the amount of memory usage in the online phase is not necessarily proportional to the number of clusters. As depicted in Figure 2.16, when considering only one cluster, 335 basis vectors are required. Evidently, this number grows to 827 by increasing the number of clusters to 30. However, when the number of clusters becomes 40, the cumulative number of basis vectors shrinks down to 564. The computer memory usage not being necessarily proportional to the number of clusters, considering a large number of clusters is a feasible option.

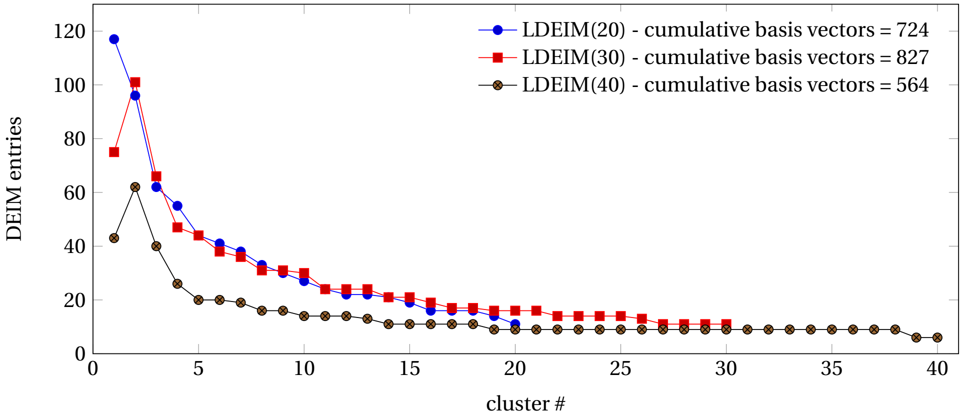


Figure 2.16: Footing: number of the DEIM entries in each cluster (reduced order model: 609 POD basis vectors associated with the displacement field).

It is also noteworthy to realize the importance of the distance metric in achieving a consistent clustering. Load-displacement curves for the 40-cluster case (black dashed line in Figure 2.15) associated with two distance metrics are depicted in Figure 2.17. It is clear that the metric based on both the Euclidean distance and the cosine similarity definition gives a proper result, whereas the metric based on the Euclidean distance alone fails to do so.

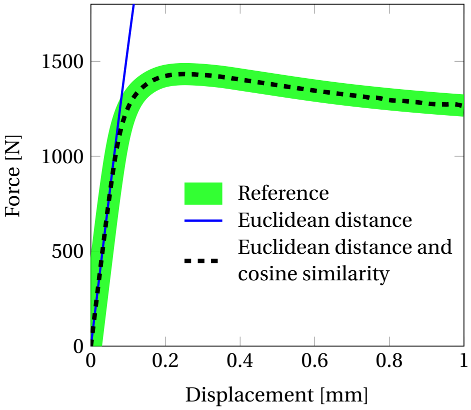


Figure 2.17: Footing: the effect of the distance definition on the load-displacement diagram (reference solution: 5184 DOFs, 1331 elements; reduced order model: 609 POD basis vectors associated with the displacement field, LDEIM with 40 clusters).

2.5. Conclusions

Considering the slow decay of the singular values of the samples collected in the offline phase, it is possible to infer that the hypothesis central to all the MOR techniques, reported in Section 3.1, is not valid for the entire loading history of the class of problems studied in this paper. To mitigate this issue, samples are divided into clusters such that for each cluster the central hypothesis becomes valid. For this purpose, a potentially computationally costly k-means clustering algorithm is added to the offline phase and, employing feature extraction, an efficient nearest neighbor classifier is added to the online phase.

We now identify a possible computational bottleneck pertinent to the reduced stiffness matrix (step 3.1.2 in Algorithm 7) –note that this is merely a technical problem and not a fundamental one as those described at the end of Section 4.2. The storage of $\boldsymbol{K} = \boldsymbol{K}^{\mathrm{L}} + \hat{\boldsymbol{K}}$ could be quite memory consuming due to the structure of the DEIM approximated nonlinear part $\hat{\boldsymbol{K}}$. Unlike the linear part of the stiffness matrix $\boldsymbol{K}^{\mathrm{L}}$, its sparsity could be considerably less than what is expected in a classical FEM analysis. Additionally, standard procedures that take advantage of the sparsity of the stiffness matrix, such as matrix assembly, might not be able to perform as efficiently as anticipated in a FEM analysis. As a result, we suggest to avoid explicitly forming $\hat{\boldsymbol{K}}$ when computing the reduced stiffness matrix by employing an appropriate matrix chain multiplication.

REFERENCES

[1] F. Ghavamian, P. Tiso, and A. Simone, *POD-DEIM model order reduction for strain-softening viscoplasticity,* Computer Methods in Applied Mechanics and Engineering **317**, 458 (2017).

[2] B. Besselink, U. Tabak, A. Lutowska, N. van de Wouw, H. Nijmeijer, D. J. Rixen, M. E. Hochstenbach, and W. H. A. Schilders, *A comparison of model reduction techniques from structural dynamics, numerical mathematics and systems and control,* Journal of Sound and Vibration **332**, 4403 (2013).

[3] C. W. Rowley, *Model reduction for fluids, using balanced proper orthogonal decomposition,* International Journal of Bifurcation and Chaos **15**, 997 (2005).

[4] K. Willcox and J. Peraire, *Balanced Model Reduction via the Proper Orthogonal Decomposition,* AIAA Journal **40**, 2323 (2002).

[5] S. Lall, J. E. Marsden, and S. Glavaški, *A subspace approach to balanced truncation for model reduction of nonlinear control systems,* International Journal of Robust and Nonlinear Control **12**, 519 (2002).

[6] S. Chaturantabut and D. C. Sorensen, *Nonlinear model reduction via discrete empirical interpolation,* SIAM Journal on Scientific Computing **32**, 2737 (2010).

[7] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. (Springer, 2002).

[8] Y. C. Liang, H. P. Lee, S. P. Lim, W. Z. Lin, K. H. Lee, and C. G. Wu, *Proper orthogonal decomposition and its applications–Part I: Theory,* Journal of Sound and Vibration **252**, 527 (2002).

[9] S. Gugercin and A. C. Antoulas, *A survey of model reduction by balanced truncation and some new results,* International Journal of Control **77**, 748 (2004).

[10] A. Astolfi, *Model reduction by moment matching for linear and nonlinear systems,* IEEE Transactions on Automatic Control **55**, 2321 (2010).

[11] P. Astrid, S. Weiland, K. Willcox, and T. Backx, *Missing point estimation in models described by proper orthogonal decomposition,* IEEE Transactions on Automatic Control **53**, 2237 (2008).

[12] K. Willcox, *Unsteady flow sensing and estimation via the gappy proper orthogonal decomposition,* Computers & Fluids **35**, 208 (2006).

[13] S. Herkt, M. Hinze, and R. Pinnau, *Convergence analysis of Galerkin POD for linear second order evolution equations,* Electronic Transactions on Numerical Analysis **40**, 321 (2013).

[14] P. Kerfriden, J. C. Passieux, and S. P. A. Bordas, *Local/global model order reduction strategy for the simulation of quasi-brittle fracture,* International Journal for Numerical Methods in Engineering **89**, 154 (2012).

[15] P. Krysl, S. Lall, and J. E. Marsden, *Dimensional model reduction in non-linear finite element dynamics of solids and structures,* International Journal for Numerical Methods in Engineering **51**, 479 (2001).

[16] A. Doostan, R. G. Ghanem, and J. Red-Horse, *Stochastic model reduction for chaos representations,* Computer Methods in Applied Mechanics and Engineering **196**, 3951 (2007).

[17] M. Chevreuil and A. Nouy, *Model order reduction based on proper generalized decomposition for the propagation of uncertainties in structural dynamics,* International Journal for Numerical Methods in Engineering **89**, 241 (2012).

[18] A. Ammar, B. Mokdad, F. Chinesta, and R. Keunings, *A new family of solvers for some classes of multidimensional partial differential equations encountered in kinetic theory modeling of complex fluids,* Journal of Non-Newtonian Fluid Mechanics **139**, 153 (2005).

[19] D. Ryckelynck, *Hyper-reduction of mechanical models involving internal variables,* International Journal for Numerical Methods in Engineering **77**, 75 (2009).

[20] P. Tiso and D. J. Rixen, *Discrete empirical interpolation method for finite element structural dynamics,* in *Topics in Nonlinear Dynamics*, Vol. 1 (Springer, 2013) pp. 203–212.

[21] P. Tiso, R. Dedden, and D. J. Rixen, *A modified discrete empirical interpolation method for reducing non-linear structural finite element models,* in *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (American Society of Mechanical Engineers, 2013).

[22] P. Perzyna, *Fundamental Problems in Viscoplasticity,* Advances in Applied Mechanics **9**, 243 (1966).

[23] B. Peherstorfer, D. Butnaru, K. Willcox, and H. J. Bungartz, *Localized discrete empirical interpolation method,* SIAM Journal on Scientific Computing **36**, A168 (2014).

[24] B. Haasdonk, M. Dihlmann, and M. Ohlberger, *A training set and multiple basis generation approach for parametrized model reduction based on adaptive grids in parameter space,* Mathematical and Computer Modelling of Dynamical Systems **17**, 423 (2011).

[25] A. Chatterjee, *An introduction to the proper orthogonal decomposition,* Current Science **78**, 808 (2000).

[26] K. Carlberg, M. Barone, and H. Antil, *Galerkin v. least-squares Petrov-Galerkin projection in nonlinear model reduction,* arXiv preprint arXiv:1504.03749v2 (2015).

[27] S. Volkwein, *Proper Orthogonal Decomposition: Theory and Reduced-Order Modeling* (University of Konstanz, 2013) lecture Notes.

[28] K. Carlberg, C. Farhat, J. Cortial, and D. Amsallem, *The GNAT method for nonlinear model reduction: Effective implementation and application to computational fluid dynamics and turbulent flows,* Journal of Computational Physics **242**, 623 (2013).

[29] A. Naik, *Data Clustering Algorithms: K-Means Clustering Algorithm,* (2010), https://sites.google.com/site/dataclusteringalgorithms/k-means-clustering-algorithm. Accessed 15 May 2016.

[30] D. Amsallem and B. Haasdonk, *PEBL-ROM: Projection-Error Based Local Reduced-Order Models,* Advanced Modeling and Simulation in Engineering Sciences **3**, 1 (2016).

[31] Q. Liang and L. Wang, *Redundancy reduction in wireless sensor networks using SVD-QR,* in *Military Communications Conference (MILCOM)* (IEEE, 2005) pp. 1857–1861.

[32] S. Chaturantabut and D. C. Sorensen, *Application of POD and DEIM on dimension reduction of nonlinear miscible viscous fingering in porous media,* Mathematical and Computer Modeling of Dynamical Systems **17**, 337 (2009).

[33] C. Farhat, P. Avery, T. Chapman, and J. Cortial, *Dimensional reduction of nonlinear finite element dynamic models with finite rotations and energy-based mesh sampling and weighting for computational efficiency,* International Journal for Numerical Methods in Engineering **98**, 625 (2014).

3

Accelerating multiscale finite element simulations of history-dependent materials using a recurrent neural network

If there is a problem you can't solve, then there is an easier problem you can solve: find it.

George Polya

FE^2 multiscale simulations of history-dependent materials are accelerated by means of a recurrent neural network (RNN) surrogate for the history-dependent micro level response. We propose a simple strategy to efficiently collect stress-strain data from the micro model, and we modify the RNN model such that it resembles a nonlinear finite element analysis procedure during training. We then implement the trained RNN model in the FE^2 scheme and employ automatic differentiation to compute the consistent tangent. The exceptional performance of the proposed model is demonstrated through a number of academic examples using strain-softening Perzyna viscoplasticity as the nonlinear material model at the micro level.

3.1. Introduction

Direct numerical simulations of heterogeneous media are often computationally intractable. To enable the simulation of these media, one can employ a computational multiscale homogenization scheme such as the multilevel finite element method (FE^2) [2]. In spite of constant advances in theoretical development and computing power, concurrent multiscale schemes are in most cases computationally too expensive to be employed in many-query applications. The main computational bottleneck in concurrent multiscale schemes is the evaluation of the micro model at each integration point in the macro

This chapter have been published in *Computer Methods in Applied Mechanics and Engineering* [1].

model. To tackle this bottleneck, we propose an efficient surrogate model for the micro model.

Since we are primarily interested in history-dependent material models for quasi-brittle failure analysis, such as Perzyna's strain-softening viscoplasticity, we choose a recurrent neural network (RNN) [3–5] as its surrogate at the micro level. An RNN model is simply a function that maps the given input sequence to the desired output sequence. To do so, it utilizes a set of history parameters. Therefore, it is inherently capable of handling history dependency. Although demonstrated on a series of academic examples in Section 3.8, this surrogate is capable of dramatically accelerating any FE^2 scheme.

To the best of our knowledge, very few surrogate models have been proposed for history-dependent constitutive laws. Among those, we refer to the work by Hambli et al. [6] who develop a feed-forward neural network as a surrogate to the damage model introduced by Chaboche [7]. Similarly, Lu et al. [8] introduce a feed-forward neural network as a surrogate for a nonlinear effective electric constitutive law [9]. A feed-forward neural network however is not capable of capturing history dependency in data.

Other than employing a surrogate model, there have been several attempts [10–13] to alleviate the aforementioned bottleneck by using dimensionality reduction modeling techniques. These contributions employ a combination of proper orthogonal decomposition (to reduce the size of the stiffness matrix) and a hyper-reduction technique (to speed up evaluation of the stiffness matrix and the internal force vector). If a model's capacity is defined as its degree of capability to learn patterns in data, the higher the capacity, the more complex patterns the model can learn. In general, the capacity of these reduced order modeling techniques is limited because they learn a linear subspace representation from high-dimensional training data.

There have been several proposals to construct data-driven frameworks for material analysis. For instance, Wang et al. [14] proposed a data-driven framework to link information between multiple scales in a recursive homogenization scheme. Bessa et al. [15] developed a self-consistent clustering analysis method to avoid the curse of dimensionality. These frameworks however cannot be implemented into an existing computer code without major intrusion.

In contrast to the previously mentioned contributions, we are interested in developing a surrogate model that has high capacity, is capable of capturing history dependency in data, and can be implemented into an existing FE^2 multiscale computer program with minimal intrusion. These objectives have been achieved thanks to the novel developments listed next. Since naively sampling the standalone micro model is computationally intractable, we propose an effective and efficient sampling strategy in Section 3.3.1. We then modify an RNN [16] model in Section 3.3.2 in such a way that it resembles a classical nonlinear finite element (FE) analysis procedure during training. In this manner the RNN model learns how to distinguish between converged and not converged data in a Newton-Raphson solution scheme. To use the RNN model in a multiscale scheme, one needs to compute the consistent tangent. This is achieved by employing automatic differentiation [17] as quickly described in Section 3.6.1—the consistent tangent is exact up to the machine precision and, using a deep learning library such as Tensorflow [18], its implementation takes exactly one line of code. And finally, we propose using the RNN in place of the history- and rate-dependent micro model to speed up the FE^2 scheme in Section 3.8. Importantly, we discuss how to boost the accuracy of the model through retraining.

3.2. PROBLEM STATEMENT

3.2.1. MULTISCALE FINITE ELEMENT ANALYSIS

To begin with, let us consider a FE^2 scheme. A FE^2 scheme consists of two models: (1) macro model in which the medium is modeled as a homogeneous material, and (2) micro model in which the medium's heterogeneity is incorporated.

MACRO MODEL

The deformation of the macro model is governed by the equilibrium equation

$$\boldsymbol{\nabla}\cdot\boldsymbol{\sigma}_{\mathrm{M}} = \mathbf{0} \quad \text{in} \quad \Omega_{\mathrm{M}}, \tag{3.1}$$

where $\boldsymbol{\nabla}$ is the differential operator, $\boldsymbol{\sigma}_{\mathrm{M}}$ is the macro stress tensor, and Ω_{M} is the volume of the macro model. For the sake of simplicity and without loss of generality we set the body force vector to zero. The volume Ω_{M} is bounded by the surface $\Gamma_{\mathrm{M}} = \Gamma^{u}_{\mathrm{M}} \cup \Gamma^{t}_{\mathrm{M}}$ on which Dirichlet (on Γ^{u}_{M}) and Neumann (on Γ^{t}_{M}) boundary conditions are applied.

The relation between the macro stress $\boldsymbol{\sigma}_{\mathrm{M}}$ and the macro strain $\boldsymbol{\varepsilon}_{\mathrm{M}}$ tensors,

$$(\boldsymbol{\sigma}_{\mathrm{M}}, \hat{\boldsymbol{h}}_{\mathrm{M}}) = \mathscr{M}(\boldsymbol{\varepsilon}_{\mathrm{M}}, \dot{\boldsymbol{\varepsilon}}_{\mathrm{M}}, \boldsymbol{h}_{\mathrm{M}}), \tag{3.2}$$

is determined through the micro model $\mathscr{M}$ discussed in the next section. In the equation above, $\boldsymbol{h}_{\mathrm{M}}$ is a set of history parameters for the macro model and $\hat{\boldsymbol{h}}_{\mathrm{M}}$ is their updated values.

Standard nonlinear finite element analysis procedures yield the weak form statement of Equation 3.1. The weak form is then discretized in time using a standard time stepping scheme. At each time step, the discretized weak form is solved using the Newton-Raphson method

$$\boldsymbol{K}_{\mathrm{M}}\,\mathrm{d}\boldsymbol{a}_{\mathrm{M}} = \boldsymbol{f}^{\mathrm{ext}}_{\mathrm{M}} - \boldsymbol{f}^{\mathrm{int}}_{\mathrm{M}}, \qquad \boldsymbol{a}_{\mathrm{M}} \leftarrow \boldsymbol{a}_{\mathrm{M}} + \mathrm{d}\boldsymbol{a}_{\mathrm{M}}. \tag{3.3}$$

In the equation above, $\boldsymbol{K}_{\mathrm{M}} = \int_{\Omega_{\mathrm{M}}} \boldsymbol{B}^{\mathrm{T}}_{\mathrm{M}}\,\boldsymbol{D}_{\mathrm{M}}\,\boldsymbol{B}_{\mathrm{M}}\,\mathrm{d}v$ is the macro stiffness matrix, $\boldsymbol{a}_{\mathrm{M}}$ is the total displacement vector of the macro model with $\mathrm{d}\boldsymbol{a}_{\mathrm{M}}$ its incremental value, $\boldsymbol{f}^{\mathrm{int}}_{\mathrm{M}} = \int_{\Omega_{\mathrm{M}}} \boldsymbol{B}^{\mathrm{T}}_{\mathrm{M}}\,\boldsymbol{\sigma}_{\mathrm{M}}\,\mathrm{d}v$ is the macro internal force vector, $\boldsymbol{f}^{\mathrm{ext}}_{\mathrm{M}} = \int_{\Gamma^{t}_{\mathrm{M}}} \boldsymbol{N}^{\mathrm{T}}_{\mathrm{M}}\,\boldsymbol{t}_{\mathrm{M}}\,\mathrm{d}s$ is the macro external force vector, $\boldsymbol{N}_{\mathrm{M}}$ is the macro model's shape function matrix, $\boldsymbol{B}_{\mathrm{M}}$ is its spatial derivative, and $\boldsymbol{D}_{\mathrm{M}}$ is the macro model's consistent tangent.

MICRO MODEL

The deformation of the micro model is also governed by the equilibrium equation

$$\boldsymbol{\nabla}\cdot\boldsymbol{\sigma}_{\mathrm{m}} = \mathbf{0} \quad \text{in} \quad \Omega_{\mathrm{m}}, \tag{3.4}$$

where $\boldsymbol{\sigma}_{\mathrm{m}}$ is the micro stress tensor, and Ω_{m} is the volume of the micro model. The volume Ω_{m} is bounded by the surface Γ_{m} on which a specific set of Dirichlet boundary conditions are applied. We specify these boundary conditions in the next section.

As an archetype of a class of history-dependent material models, we choose a Perzyna viscoplasticity similar to the one employed in Reference [19]. The stress is related to the strain through the constitutive relation expressed in rate form

$$\dot{\boldsymbol{\sigma}}_{\mathrm{m}} = \boldsymbol{D}^{\mathrm{e}}\,(\dot{\boldsymbol{\varepsilon}}_{\mathrm{m}} - \dot{\boldsymbol{\varepsilon}}_{\mathrm{m}}^{\mathrm{vp}}), \tag{3.5}$$

where $\dot{\#}$ is time derivative of #, $\boldsymbol{D}^{\mathrm{e}}$ is the elastic modulus tensor, $\boldsymbol{\varepsilon}_{\mathrm{m}}$ is the total micro strain tensor, and $\boldsymbol{\varepsilon}_{\mathrm{m}}^{\mathrm{vp}}$ is the micro viscoplastic strain tensor.

The micro viscoplastic strain grows at the rate of

$$\dot{\boldsymbol{\varepsilon}}_{\mathrm{m}}^{\mathrm{vp}} = \dot{\lambda}\,\frac{\partial\theta}{\partial\boldsymbol{\sigma}_{\mathrm{m}}} \tag{3.6}$$

where $\dot{\lambda} = \eta < \phi >^{\beta}$ is the rate of plastic multiplier, <> is the Macaulay bracket, η is the viscosity parameter, β is a model parameter, $\phi = \frac{\theta}{\sigma_{\mathrm{Y}}}$ is the overstress function, $\theta = \sigma_{\mathrm{vm}} - \sigma_{\mathrm{Y}}$ is the yield function, σ_{vm} is the von Mises stress, and σ_{Y} is the current yield stress.

To introduce a strain softening response we decrease the yield stress as the accumulated plastic strain κ increases through

$$\sigma_{\mathrm{Y}} = \sigma_{\mathrm{Y}_0}((1+a)\,e^{-b\kappa} - a\,e^{-2b\kappa}) \tag{3.7}$$

where a and b are model parameters, σ_{Y_0} is the initial yield stress, and the accumulated plastic strain $\kappa = \lambda$.

Standard nonlinear finite element analysis procedures yield the weak form of Equation 3.4. At each time step and Newton-Raphson iteration for the macro model, and in each integration point of the macro model, the weak form of Equation 3.4 is solved using the Newton-Raphson method

$$\boldsymbol{K}_{\mathrm{m}}\,\mathrm{d}\boldsymbol{a}_{\mathrm{m}} = -\boldsymbol{f}_{\mathrm{m}}^{\mathrm{int}}, \qquad \boldsymbol{a}_{\mathrm{m}} \leftarrow \boldsymbol{a}_{\mathrm{m}} + \mathrm{d}\boldsymbol{a}_{\mathrm{m}}. \tag{3.8}$$

In the equation above $\boldsymbol{K}_{\mathrm{m}} = \int_{\Omega_{\mathrm{m}}} \boldsymbol{B}_{\mathrm{m}}^{\mathrm{T}}\,\boldsymbol{D}_{\mathrm{m}}\,\boldsymbol{B}_{\mathrm{m}}\,\mathrm{d}v$ is the micro stiffness matrix, $\boldsymbol{a}_{\mathrm{m}}$ is the micro total displacement vector and $\mathrm{d}\boldsymbol{a}_{\mathrm{m}}$ is its incremental value, $\boldsymbol{f}_{\mathrm{m}}^{\mathrm{int}} = \int_{\Omega_{\mathrm{m}}} \boldsymbol{B}_{\mathrm{m}}^{\mathrm{T}}\,\boldsymbol{\sigma}_{\mathrm{m}}\,\mathrm{d}v$ is the micro internal force vector, $\boldsymbol{N}_{\mathrm{m}}$ is the micro shape function matrix, $\boldsymbol{B}_{\mathrm{m}}$ is its spatial derivative, and $\boldsymbol{D}_{\mathrm{m}}$ is the micro model's consistent tangent.

3.2.2. Micro to macro

At each integration point in the macro model, a strain tensor is passed to the micro model. Considering first-order homogenization, the principle of separation of scales, and the zero boundary fluctuations model [20], the macroscopic strain is imposed on the boundary Γ_{m} of the micro model via

$$\boldsymbol{a}_{\mathrm{m}} = \boldsymbol{\varepsilon}_{\mathrm{M}}\,\Delta\boldsymbol{x}, \tag{3.9}$$

where $\Delta\boldsymbol{x}$ is the relative position vector on the micro model. The micro model is then solved as a boundary value problem.

To compute the macro stress tensor $\boldsymbol{\sigma}_\mathrm{M}$, the Hill-Mandel principle is used. As a consequence of this principle, the macro stress tensor becomes the volumetric average of the work done on the surface boundary of the micro model:

$$\boldsymbol{\sigma}_\mathrm{M} = \frac{1}{\Omega_\mathrm{m}} \sum_{i=1}^{n} \boldsymbol{f}_\mathrm{m}^{i} \otimes \boldsymbol{u}_\mathrm{m}^{i}, \tag{3.10}$$

where the sum is over the n boundary nodes of the micro model. The vectors $\boldsymbol{f}_\mathrm{m}^{i}$ and $\boldsymbol{u}_\mathrm{m}^{i}$ are, respectively, the nodal internal force and displacement vectors of the micro model's boundary node i.

The macro-scale consistent tangent $\boldsymbol{D}_\mathrm{M} = \frac{\partial \boldsymbol{\sigma}_\mathrm{M}}{\partial \boldsymbol{\varepsilon}_\mathrm{M}}$ is computed using the probing method. Details of the algorithm can be found in Reference [21, Box 2].

3.2.3. COMPUTATIONAL BOTTLENECK

The computational bottleneck in a multiscale scheme is due to the evaluation of the micro model for each integration point of the macro model: specifically, the solution of the system of equations (3.8), and the evaluation of the stiffness matrix $\boldsymbol{K}_\mathrm{m}$ and the internal force vector $\boldsymbol{f}_\mathrm{m}^\mathrm{int}$.

In principle, there are two possible approaches to overcome this bottleneck: (1) a reduction of the computational cost related to the solution of (3.8) and the evaluation of $\boldsymbol{K}_\mathrm{m}$ and $\boldsymbol{f}_\mathrm{m}^\mathrm{int}$, and (2) a complete replacement of the micro model with a computationally efficient surrogate. In this paper we consider the latter approach.

The rest of the paper is dedicated to the development of a surrogate model that serves as a computationally efficient surrogate for the micro model.

3.3. SURROGATE FOR THE HISTORY-DEPENDENT MICRO MODEL

Before going any further, let us define the notation used in this work. First of all, for the sake of readability, we refer to the macro strain and stress tensors as simply strain and stress tensors. We also drop the subscript M from the symbols of such quantities. We then indicate the i-th sequence by superscript (i), the s-th step in a sequence by superscript $[s]$. For example, we refer to the strain tensor at step s as $\boldsymbol{\varepsilon}^{[s]}$, a strain tensor sequence as $\boldsymbol{E} = (\boldsymbol{\varepsilon}^{[1]}, ..., \boldsymbol{\varepsilon}^{[s]} ..., \boldsymbol{\varepsilon}^{[S]})$, the i-th strain tensor sequence sample as $\boldsymbol{E}^{(i)}$, and the i-th strain tensor sequence sample and its corresponding stress tensor sequence as $\{\boldsymbol{E}, \boldsymbol{\Sigma}\}^{(i)}$.

In a nonlinear FEM scheme, the micro model from Section 3.2.1 maps a sequence of strain tensors $\boldsymbol{E} = (\boldsymbol{\varepsilon}^{[1]}, ..., \boldsymbol{\varepsilon}^{[S]})$ to a sequence of stress tensors $\boldsymbol{\Sigma} = (\boldsymbol{\sigma}^{[1]}, ..., \boldsymbol{\sigma}^{[S]})$:

$$\mathscr{M} : \boldsymbol{E} \to \boldsymbol{\Sigma}. \tag{3.11}$$

Index s is the accumulated Newton-Raphson iteration number—$s = 1$ corresponds to the first time step and the first Newton-Raphson iteration, while $s = S$ corresponds to the last Newton-Raphson iteration of the last time step.

The aforementioned mapping is the computational bottleneck. To tackle this bottleneck, we introduce a surrogate

$$\mathscr{D} : \boldsymbol{E} \to \hat{\boldsymbol{\Sigma}} \tag{3.12}$$

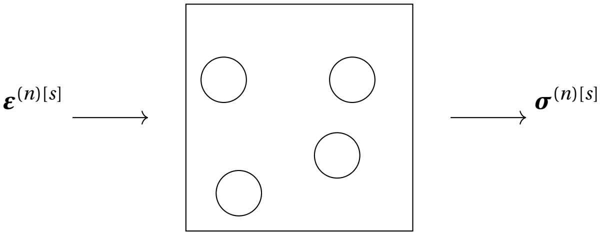

$\boldsymbol{\varepsilon}^{(n)[s]}$
$\boldsymbol{\sigma}^{(n)[s]}$

Figure 3.1: Collecting strain sequence $\boldsymbol{E}^{(n)}$ and its corresponding strain sequence $\boldsymbol{\Sigma}^{(n)}$ from the standalone micro model. This is a computationally intractable task. The space of all possible strain sequences is extremely large and sampling is computationally intractable.

for the micro model $\mathscr{M}$ where $\hat{\boldsymbol{\Sigma}}$ is the output of the surrogate model and essentially an approximation of $\boldsymbol{\Sigma}$.

The surrogate model $\mathscr{D}$ contains a number of parameters $\boldsymbol{W}$. These parameters need to be tuned such that the predictions of the surrogate become accurate. Concretely, the parameters $\boldsymbol{W}$ are tuned such that the discrepancy

$$e = \|\hat{\boldsymbol{\Sigma}} - \boldsymbol{\Sigma}\|_2 \tag{3.13}$$

between the predictions of the surrogate $\hat{\boldsymbol{\Sigma}}$ and the micro model outputs $\boldsymbol{\Sigma}$ is small. This process, discussed in Section 4.7, is referred to as training.

To train the surrogate, (3.13) clearly tells us that we not only require a strain tensor sequence $\boldsymbol{E}$ to compute the surrogate's prediction $\hat{\boldsymbol{\Sigma}}$, but we also require the corresponding stress tensor sequence $\boldsymbol{\Sigma}$ from the micro model. The collection of these data is referred to as sampling. We further discuss our sampling strategy in the next section.

3.3.1. SAMPLING

To train the surrogate, one needs to collect a set of strain tensor sequences and their corresponding stress tensor sequences from the micro model.

Naively, as depicted in Figure 3.1, one could apply several different strain sequences $\boldsymbol{E}^{(n)}$ to the standalone micro model, compute the corresponding stress sequences $\boldsymbol{\Sigma}^{(n)}$, and then collect the data $\{\boldsymbol{E}, \boldsymbol{\Sigma}\}^{(n)}$, where $n = 1, \ldots, N$ and N is the total number of samples. In practice however this exercise is not technically viable. The space of all possible strain sequences is extremely large and sampling is therefore computationally intractable.

Due to the problem stated above, we employ a different strategy to collect the strain-stress data. We essentially consider only a subspace of all possible strain sequences. Specifically, in contrast with the naive approach, we assume that the data is generated from a set of macro models (the strategy is detailed in the next paragraph). In this manner, we only sample a small part of the space of all possible strain sequences. Hence, the sampling procedure becomes computationally tractable. The downside, however, is that the surrogate model that is trained on these data can only perform accurately in the macro models that generated the data.

Let us concretely define our sampling strategy: we consider a set of macro models with a set of configurations (boundary conditions, geometrical features, ...); we run the

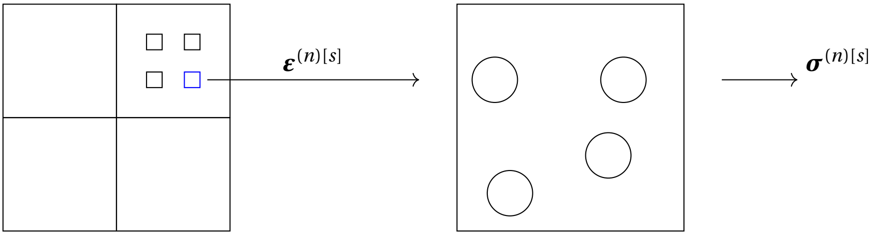

$\boldsymbol{\varepsilon}^{(n)[s]}$
$\boldsymbol{\sigma}^{(n)[s]}$

Figure 3.2: Collecting strain sequence $\boldsymbol{E}^{(n)}$ and its corresponding strain sequence $\boldsymbol{\Sigma}^{(n)}$. In contrast to the strategy depicted in Figure 3.1, here we collect these samples at integration points of an assumed macro model. This is a computationally tractable task. However, the surrogate that is trained on this data set only performs accurately when used in the assumed macro model.

FEM simulation, and at each integration point in the macro model we impose the macro strain $\boldsymbol{\varepsilon}^{[s]}$ on the micro model and compute the macro stress $\boldsymbol{\sigma}^{[s]}$; at the end of the simulation we stack the macro strain tensors to form $\boldsymbol{E}^{(n)}$ and the macro stress tensors to form $\boldsymbol{\Sigma}^{(n)}$ and collect $\{\boldsymbol{E},\boldsymbol{\Sigma}\}^{(n)}$. This procedure is repeated for several sets of macro model's configurations. Note that n unequivocally identifies a certain macro model, a certain configuration, and a certain integration point of the macro model.

The sampling strategy is depicted in Figure 3.2 and the data structure is depicted in Table 3.1.

In nonlinear FE analysis

time step: 1	
Newton-Raphson iteration: 1	s=1
Newton-Raphson iteration: 2	s=2
Newton-Raphson iteration: 3	s=3
time step: 2	
Newton-Raphson iteration: 1	s=4
time step: 3	
Newton-Raphson iteration: 1	s=5
Newton-Raphson iteration: 2	s=6

Sample data from an integration point

$$\boldsymbol{E}^{(n)} = (\boldsymbol{\varepsilon}^{(n)[1]}, \boldsymbol{\varepsilon}^{(n)[2]}, \boldsymbol{\varepsilon}^{(n)[3]}, \boldsymbol{\varepsilon}^{(n)[4]}, \boldsymbol{\varepsilon}^{(n)[5]}, \boldsymbol{\varepsilon}^{(n)[6]})$$

$$\boldsymbol{\Sigma}^{(n)} = (\boldsymbol{\sigma}^{(n)[1]}, \boldsymbol{\sigma}^{(n)[2]}, \boldsymbol{\sigma}^{(n)[3]}, \boldsymbol{\sigma}^{(n)[4]}, \boldsymbol{\sigma}^{(n)[5]}, \boldsymbol{\sigma}^{(n)[6]})$$

$$\boldsymbol{b}^{(n)} = (b^{(n)[1]}, b^{(n)[2]}, b^{(n)[3]}, b^{(n)[4]}, b^{(n)[5]}, b^{(n)[6]})$$

Table 3.1: Data collection method: data is collected at each Newton-Raphson iteration.

The data that we collect from the macro model, at each step s, could correspond to a converged or a not converged Newton-Raphson iteration. We keep track of converged Newton-Raphson iteration through the scalar $b^{[s]}$ and refer to it as the convergence indicator. We set $b^{[s]}$ to one if the data at index s is converged or otherwise set it to zero. We then collect these scalars in the convergence indicator sequence $\boldsymbol{b} = (b^{[1]}, ..., b^{[S]})$. Later, in Section 3.3.2, $\boldsymbol{b}$ is used to treat converged and not converged data differently.

It is important that the data are collected from all Newton-Raphson iterations, converged and not converged. To illustrate this matter, consider a scenario in which we

iterations	previous solution	current solution
1	a^{t-1}	a_1
2	a_1	a_2
3	a_2	a_3

Table 3.2: Newton-Raphson iterations of time step t.

train the surrogate only on the converged data. As a result, the surrogate would perform poorly at the intermediate Newton-Raphson iterations. As an illustrative example, consider the Newton-Raphson iterations in Table 3.2. Let us consider the scenario in which the surrogate model is only trained on the data from the converged Newton-Raphson iterations. At time step t, the Newton-Raphson scheme uses the converged solution at time step $t-1$ to compute the solution of the first iteration a_1, but it does so inaccurately. Then it uses the solution of the first Newton-Raphson iteration a_1 to compute that of the second a_2. The solution of the first Newton-Raphson iteration was already erroneous, therefore the solution of the second iteration can only be worse. In this fashion, the error accumulates in the following Newton-Raphson iterations. Training the surrogate on both converged and not converged steps circumvents this issue. When dealing with a nonlinear problem, it is common practice to train the model on both converged and not converged data as done for instance in Refereces [22, 23]. In practice, we observe that the models that were trained on converged and not converged data are often numerically more stable than those that are only trained on converged data.

To summarize, using the sampling strategy depicted in Figure 3.2, we collect samples $\{\boldsymbol{E}, \boldsymbol{\Sigma}, \boldsymbol{b}\}^{(n)}$, where $n = 1, ..., N$ and N is the total number of sequences.

3.3.2. Recurrent neural network

We employ a recurrent neural network (RNN), which is a class of deep learning models, to map the strain sequence $\boldsymbol{E} = (\boldsymbol{\varepsilon}^{[1]}, ..., \boldsymbol{\varepsilon}^{[S]})$ and convegence indicator sequence $\boldsymbol{b} = (b^{[1]}, ..., b^{[S]})$ to stress sequence $\boldsymbol{\Sigma} = (\boldsymbol{\sigma}^{[1]}, ..., \boldsymbol{\sigma}^{[S]})$. Different types of RNN models are discussed in Reference [16]. Here, we propose the RNN model in Figure 3.3, which essentially is a computational graph. It consists of several nodes (mLSTM, dropout, and dense cells) that represent operations and a number of edges that represent operands ($\boldsymbol{\varepsilon}^{[s]}$, $\boldsymbol{h}^{[s]}$, $\boldsymbol{z}^{[s]}$, $\tilde{\boldsymbol{z}}^{[s]}$, $\boldsymbol{\sigma}^{[s]}$).

Let us break down the RNN graph in Figure 3.3 by walking through it. The graph takes vectorized versions of strain $\boldsymbol{\varepsilon}$ and stress $\boldsymbol{\sigma}$ tensors. Then, at each step s, the modified Long-Short Term Memory (mLSTM) cell, further elaborated in Section 3.3.2, maps the strain vector $\boldsymbol{\varepsilon}^{[s]}$, the convergence indicator $b^{[s]}$, and the history vector $\boldsymbol{h}^{[s-1]}$ to the vector $\boldsymbol{z}^{[s]}$ and the updated set of history vectors $\boldsymbol{h}^{[s]}$. Note that $\boldsymbol{z}^{[s]}$ depends not only on $\boldsymbol{\varepsilon}^{[s]}$ but also on $(\boldsymbol{\varepsilon}^{[1]}, ..., \boldsymbol{\varepsilon}^{[s-1]})$ through the set of history vectors $\boldsymbol{h}^{[s-1]}$. This attribute resembles a history-dependent constitutive model. The output of the mLSTM cell $\boldsymbol{z}^{[s]}$ is then passed through the dropout cell [24]. The dropout cell sets some of $\boldsymbol{z}^{[s]}$ entries to zero and outputs $\tilde{\boldsymbol{z}}^{[s]}$. How dropout works and its merits are further discussed in Section 3.3.2. Finally, the dense cell, which is described in Section 3.3.2, maps $\tilde{\boldsymbol{z}}^{[s]}$ to the output stress vector $\hat{\boldsymbol{\sigma}}^{[s]}$.

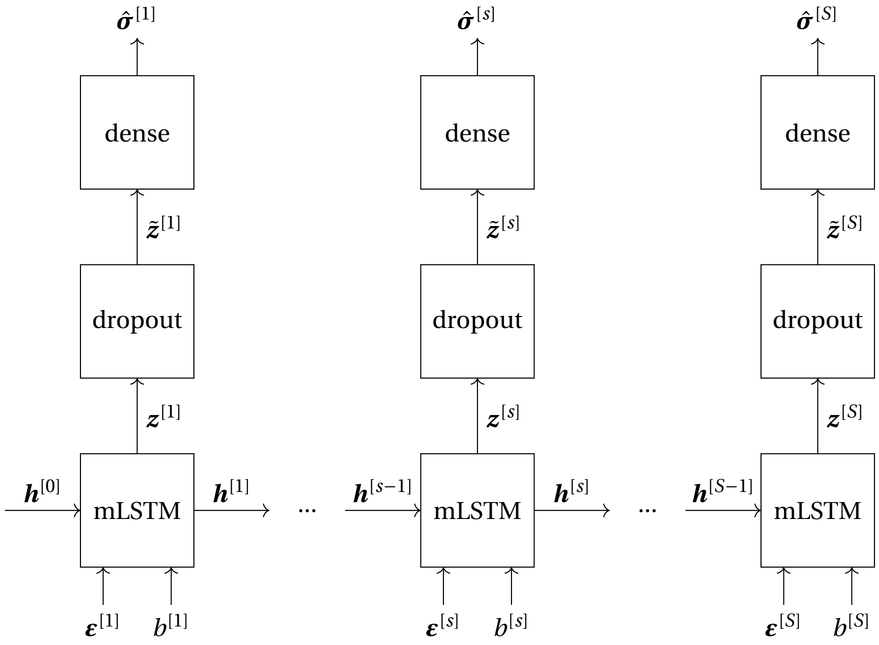

$\hat{\boldsymbol{\sigma}}^{[1]}$
$\hat{\boldsymbol{\sigma}}^{[s]}$
$\hat{\boldsymbol{\sigma}}^{[S]}$
dense
dense
dense
$\tilde{\boldsymbol{z}}^{[1]}$
$\tilde{\boldsymbol{z}}^{[s]}$
$\tilde{\boldsymbol{z}}^{[S]}$
dropout
dropout
dropout
$\boldsymbol{z}^{[1]}$
$\boldsymbol{z}^{[s]}$
$\boldsymbol{z}^{[S]}$
$\boldsymbol{h}^{[0]}$
mLSTM
$\boldsymbol{h}^{[1]}$
…
$\boldsymbol{h}^{[s-1]}$
mLSTM
$\boldsymbol{h}^{[s]}$
…
$\boldsymbol{h}^{[S-1]}$
mLSTM
$\boldsymbol{\varepsilon}^{[1]}$
$b^{[1]}$
$\boldsymbol{\varepsilon}^{[s]}$
$b^{[s]}$
$\boldsymbol{\varepsilon}^{[S]}$
$b^{[S]}$

Figure 3.3: Recurrent neural network computational graph. It consists of several nodes (mLSTM, dropout, and dense cells) that represent operations and a number of edges that represent operands (the strain vector $\boldsymbol{\varepsilon}^{[s]}$, a set of history vectors $\boldsymbol{h}^{[s]}$, the output of the mLSTM cell $\boldsymbol{z}^{[s]}$, the output of the dropout cell $\hat{\boldsymbol{z}}^{[s]}$, and the output stress vector $\boldsymbol{\sigma}^{[s]}$)

In the following sections we elaborate on the mLSTM cell, discuss dropout and its merits, and define the mapping from $\boldsymbol{z}^{[s]}$ to $\hat{\boldsymbol{\sigma}}^{[s]}$ in the dense cell.

MODIFIED LONG-SHORT TERM MEMORY WHICH RESEMBLE A NONLINEAR FINITE ELEMENT ANALYSIS

Recurrent neural networks are known to suffer from vanishing gradients. The vanishing gradient problem manifests itself while training RNN models with gradient-based optimizers (refer to Hochreiter et al. [25, 26] for a detailed description). The consequence of vanishing gradients is that the training takes too much time. To overcome this problem, Hochreiter and Schmidhuber [27] suggest employing a specific cell called Long-Short Term Memory (LSTM). A clear explanation of the LSTM cell is given by Olah [28]; here, for the sake of completeness, we explain its algorithm. Note that, except for our modification, we follow the standard formulation of the LSTM cell. Furthermore, it should be noted that one should avoid endowing operands or operations in the LSTM cell with any physical semantics as they are replaceable with other operands and operations, as done for instance by Cho et al. [29].

A typical LSTM cell accepts an input vector $\boldsymbol{\varepsilon}^{[s]}$ and a set of history vectors $\boldsymbol{h}^{[s-1]} = \{\boldsymbol{d}^{[s-1]}, \boldsymbol{c}^{[s-1]}\}$. The history vectors are commonly referred to as hidden state $\boldsymbol{d}^{[s-1]} \in \mathbb{R}^k$ and cell state $\boldsymbol{c}^{[s-1]} \in \mathbb{R}^k$, where k is the size of the LSTM cell. After processing the input and history vectors, the LSTM cell produces the output $\boldsymbol{z}^{[s]}$ and updates the history vectors $\boldsymbol{h}^{[s]}$.

We modify a typical LSTM cell such that it resembles the stress update procedure in a typical nonlinear FE analysis with a history-dependent constitutive model. We refer to the modified LSTM cell as mLSTM. In such FE analysis, only the history parameters of the converged Newton-Raphson step is passed to the next time step. In the mLSTM cell, we enforce such behavior by only allowing the history vectors of the converged steps to pass on. To this end, we modify the history vectors $\boldsymbol{h}^{[s]}$ emission mechanism. We pass in an additional input $b^{[s]}$ to the LSTM cell. If the value of $b^{[s]}$ is equal to one, we emit the updated history vectors $\boldsymbol{h}^{[s]}$. Otherwise, we reset the history vectors to that of the previous steps and emit them. In this manner, we only pass the history vectors of the converged Newton-Raphson steps to the next step. Next, we describe the overall mechanism of an mLSTM cell.

The mLSTM cell starts with updating the cell state

$$\boldsymbol{c}^{[s]} = \boldsymbol{f}^{[s]} \odot \boldsymbol{c}^{[s-1]} + \boldsymbol{i}^{[s]} \odot \hat{\boldsymbol{c}}^{[s]}, \tag{3.14}$$

then it updates the hidden state

$$\boldsymbol{d}^{[s]} = \boldsymbol{o}^{[s]} \odot \tanh(\boldsymbol{c}^{[s]}), \tag{3.15}$$

and finally it computes the output vector

$$\boldsymbol{z}^{[s]} = \boldsymbol{o}^{[s]} \odot \tanh(\boldsymbol{c}^{[s]}). \tag{3.16}$$

The modification that we introduce is that we only update the history vectors $\boldsymbol{h}^{[s]} = \{\boldsymbol{d}^{[s]}, \boldsymbol{c}^{[s]}\}$ if the value of $b^{[s]}$ is one. If b is equal zero, we keep using the previous history vectors $\boldsymbol{h}^{[s-1]}$.

In the equations above, $\odot$ is the element-wise product operation, vectors

$$\boldsymbol{i}^{[s]} = \text{sigmoid}(\boldsymbol{W}_e^i \boldsymbol{\varepsilon}^{[s]} + \boldsymbol{W}_d^i \boldsymbol{d}^{[s-1]} + \boldsymbol{b}^i), \tag{3.17}$$

$$\boldsymbol{f}^{[s]} = \text{sigmoid}(\boldsymbol{W}_e^f \boldsymbol{\varepsilon}^{[s]} + \boldsymbol{W}_d^f \boldsymbol{d}^{[s-1]} + \boldsymbol{b}^f), \tag{3.18}$$

$$\boldsymbol{o}^{[s]} = \text{sigmoid}(\boldsymbol{W}_e^o \boldsymbol{\varepsilon}^{[s]} + \boldsymbol{W}_d^o \boldsymbol{d}^{[s-1]} + \boldsymbol{b}^o), \tag{3.19}$$

are the input, the forget, and the output gates, respectively, vector $\boldsymbol{c}^{[s-1]}$ is the previous step's cell state, and vector

$$\hat{\boldsymbol{c}}^{[s]} = \tanh(\boldsymbol{W}_e^c \boldsymbol{\varepsilon}^{[s]} + \boldsymbol{W}_d^c \boldsymbol{d}^{[s-1]} + \boldsymbol{b}^c), \tag{3.20}$$

is the candidate for the updated cell state. The tanh and sigmoid functions are meant to work element-wise across a vector.

In the mLSTM cell, there are in total four sets of trainable weight matrices $\boldsymbol{W}_e^\# \in \mathbb{R}^{k\times l}$, $\boldsymbol{W}_d^\# \in \mathbb{R}^{k\times k}$, and bias vectors $\boldsymbol{b}^\# \in \mathbb{R}^k$, where $\# \in \{i, o, c, f\}$, k is the size of the mLSTM cell, and l is the size of the strain and stress vectors. We discuss how to tune these trainable parameters in Section 4.7.

The gates take values between zero and one and therefore they serve as filters. At one extreme, if the input gate is zero and the forget gate is one, the model completely ignores the history. And at another extreme, if the input gate is one and the forget gate is zero, the model only uses the history and ignores the current step's computations. This attribute endows the model with the capability of capturing very long term dependencies and simultaneously the ability of completely ignoring even very short term dependencies.

REGULARIZATION VIA DROPOUT

We expect an RNN model not only to fit to the data that it is trained on, but also to learn patterns in the data and therefore become capable of making accurate predictions on the new data. This is however challenging. By simply increasing the size of the mLSTM cell, the model overfits to the training data and fails to generalize. The standard way to mitigate overfitting is to regularize the model. In this section we introduce an effective regularization technique called dropout [24].

During training, the dropout cell takes the output of the mLSTM cell $\boldsymbol{z}^{[s]}$ and randomly zeros some of its entries through

$$\hat{\boldsymbol{z}}^{[s]} = \frac{1}{p}\boldsymbol{r} \odot \boldsymbol{z}^{[s]}, \tag{3.21}$$

where $\boldsymbol{r}$ is a vector of Bernoulli random numbers with probability $p \in (0,1]$ where p is the probability that each entry of $\boldsymbol{z}^{[s]}$ is kept. In other words, p tunes the level of reguralization—the lower the value of p, the higher the effect of reguralization. After training is over, and when the model is used, the value of p is set to one, and the whole network is used for prediction. Since p is less than one during training, it is possible to to show that the average value of $\hat{\boldsymbol{z}}^{[s]}$ becomes larger than $\boldsymbol{z}^{[s]}$ [24]. To remedy this issue, the right-hand side of (3.21) is multiplied by $1/p$ [30].

Training is an iterative procedure and is discussed further in Section 4.7. At each training step, the dropout makes a part of the network inactive. In other words, it is as if a smaller network is being trained at each training step. This can be regarded as ensemble learning [24], an approach that regularizes the model and increase its generalization capabilities.

COMPUTING THE PREDICTION OF THE MODEL

At the very top of the RNN model, in the dense cell, we map the output of the dropout $\hat{\boldsymbol{z}}^{[s]}$ to the model's prediction of the stress vector according to

$$\hat{\boldsymbol{\sigma}}^{[s]} = \boldsymbol{W}\tilde{\boldsymbol{z}}^{[s]} + \boldsymbol{b}, \tag{3.22}$$

where the weight matrix $\boldsymbol{W} \in \mathbb{R}^{l\times k}$ and the bias vector $\boldsymbol{b} \in \mathbb{R}^{l}$ contain trainable parameters, the size of the stress vector is l, and the size of the dense layer k is equal to the size of the output of the mLSTM cell.

The sole purpose of the linear transformation is to map the output of the dropout cell from the space of $\hat{\boldsymbol{z}}^{[s]}$ to the space of $\hat{\boldsymbol{\sigma}}^{[s]}$.

3.4. TRAINING

Before training, the model predictions are erroneous. We employ a loss function to measure the error e between the predicted stress vector sequence $\hat{\boldsymbol{\Sigma}}$ and the sampled stress vector sequence $\boldsymbol{\Sigma}$. We further discuss the loss function in Section 3.4.1.

In Sections 3.3.2 and 3.3.2 we indicated the trainable parameters. The training consists in the optimization of these parameters such that the aforementioned error becomes sufficiently small. The optimization technique that we employed is introduced in Section 4.5.3. The training graph of the RNN model is depicted in Figure 3.4. The graph

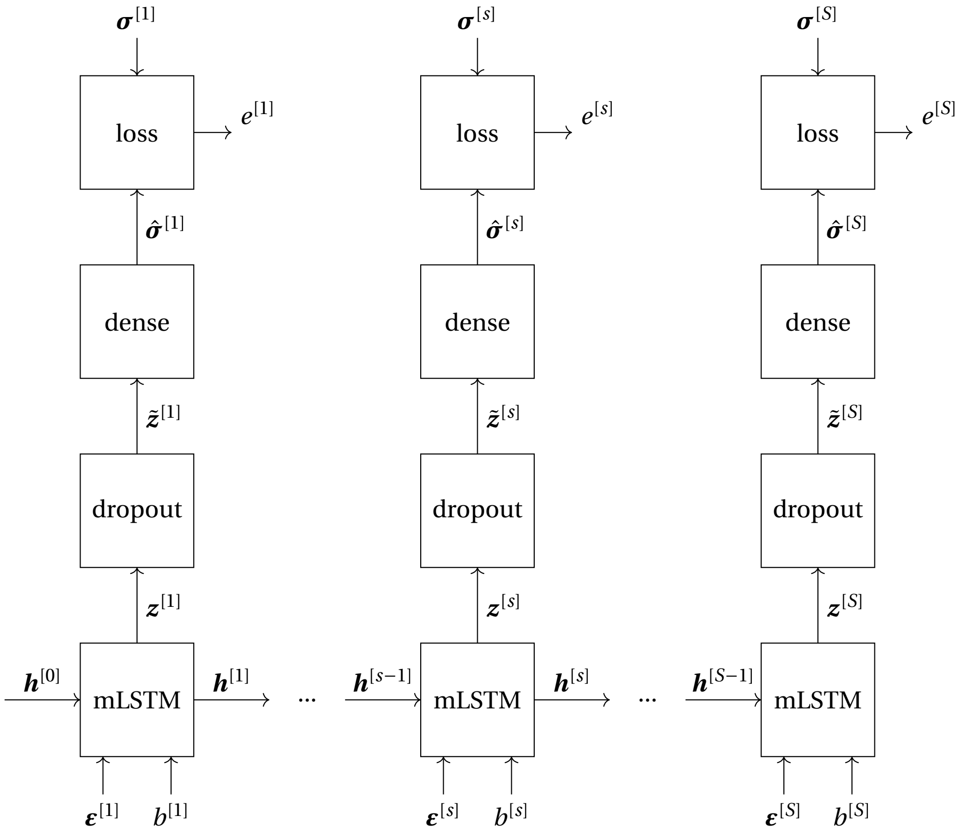

$\boldsymbol{\sigma}^{[1]}$
$\boldsymbol{\sigma}^{[s]}$
$\boldsymbol{\sigma}^{[S]}$
loss
$e^{[1]}$
$e^{[s]}$
$e^{[S]}$
$\hat{\boldsymbol{\sigma}}^{[1]}$
$\hat{\boldsymbol{\sigma}}^{[s]}$
$\hat{\boldsymbol{\sigma}}^{[S]}$
dense
$\tilde{\boldsymbol{z}}^{[1]}$
$\tilde{\boldsymbol{z}}^{[s]}$
$\tilde{\boldsymbol{z}}^{[S]}$
dropout
$\boldsymbol{z}^{[1]}$
$\boldsymbol{z}^{[s]}$
$\boldsymbol{z}^{[S]}$
$\boldsymbol{h}^{[0]}$
mLSTM
$\boldsymbol{h}^{[1]}$
...
$\boldsymbol{h}^{[s-1]}$
$\boldsymbol{h}^{[s]}$
$\boldsymbol{h}^{[S-1]}$
$\boldsymbol{\varepsilon}^{[1]}$
$b^{[1]}$
$\boldsymbol{\varepsilon}^{[s]}$
$b^{[s]}$
$\boldsymbol{\varepsilon}^{[S]}$
$b^{[S]}$

Figure 3.4: The training graph of the RNN model. The graph in Figure 3.4 is essentially the graph in Figure 3.3 plus the additional node on top, which we refer to as loss cell. The loss cell takes the output stress vector $\hat{\boldsymbol{\sigma}}$ together with the target stress vector $\boldsymbol{\sigma}$ and computes the error e using the loss function (3.23).

in Figure 3.4 is essentially the graph in Figure 3.3 plus the additional node on top, which we refer to as the loss cell. The loss cell takes the output stress vector $\hat{\boldsymbol{\sigma}}$ together with the target stress vector $\boldsymbol{\sigma}$ and computes the error e using the loss function (3.23).

3.4.1. Loss function

We measure the difference between the output of the RNN model and the target output by computing the mean squared error

$$e = \frac{1}{N}\sum_{n=1}^{N} E^{(n)} \quad \text{with} \quad E^{(n)} = \frac{1}{S^{(n)}}\sum_{s=1}^{S^{(n)}} e^{[s]} \quad \text{and} \quad e^{[s]} = \|\boldsymbol{\sigma}^{[s]} - \hat{\boldsymbol{\sigma}}^{[s]}\|_2^2, \tag{3.23}$$

where N is the total number of training data sets and $S^{(n)}$ is the length of the n-th sequence. It is important to normalize the error with N and $S^{(n)}$ because one can then compare the magnitude of error across several models with different number of training data and sequence lengths.

3.4.2. OPTIMIZATION OF TRAINABLE PARAMETERS

Trainable parameters in a deep learning model are commonly optimized using a gradient-based optimizer. These parameters are initially set to random numbers and then are iteratively tuned until the value of the error (3.23) becomes small. To this end, we use a specific gradient-based optimizer called Adam [31]. For further reference, Goh [32] explains how optimizers that include momentum, such as Adam, should be understood in an amusingly interactive manner.

3.5. HYPERPARAMETERS

There are several parameters in the RNN model that we cannot train using a gradient-based optimizer. These are commonly referred to as hyperparameters. These parameters, however, have impact on the training process and performance of the trained model and therefore we need to tune them. The hyperparameters that we tune are the size of the mLSTM cell k and the dropout probability p. We consider typical values for the rest of the hyperparameters. For instance, we accept the additional training cost and do not truncate the input-output sequences, and we employ the recommendation of Kingma et al. [31] for the parameters of the Adam optimizer.

Tuning these hyperparameters is essentially a trial-and-error exercise. We utilize a mixture of manual trial-and-error attempts and Bayesian optimization method [33] to tune them.

Due to the bias and variance trade-off [34], one should not expect the generalization capability of the RNN model to monotonically increase by, for instance, increasing the mLSTM cell size k. This is in contrast to a finite element analysis in which, for instance, by increasing the resolution of the mesh the quality of the solution monotonically increases.

3.6. SURROGATE INJECTION INTO THE MACRO MODEL

Up to this point the RNN model that is meant to serve as a surrogate for the micro model is fully discussed. In this section, we discuss the injection of this surrogate into the macro model.

Let us first consider the stress update procedure in the macro model. Figure 3.5 depicts this procedure. In each integration point, at each time step t, and Newton-Raphson iteration i, we pass a strain tensor $\boldsymbol{\varepsilon}^{[t,i]}$ and a strain increment $\Delta\boldsymbol{\varepsilon}^{[t,i]} = \boldsymbol{\varepsilon}^{[t,i]} - \boldsymbol{\varepsilon}^{[t,i-1]}$ to the micro model. We apply strain tensors to the micro model as a set of Dirichlet boundary conditions, as defined in Section 3.8. We then evaluate the micro model and compute the stress $\boldsymbol{\sigma}^{[t,i]}$ and a set of history parameters. Should the Newton-Raphson scheme converge, we update the history parameters $\boldsymbol{h}_{\mathrm{m}}^{[t]}$.

Let us inject the RNN model in the aforementioned stress update procedure. The stress update procedure with the RNN model in place of the micro model is depicted in Figure 3.6. There are a number of differences between the computational graph in chapter-3/figures 3.5 and 3.6. First, the RNN model does not need the incremental

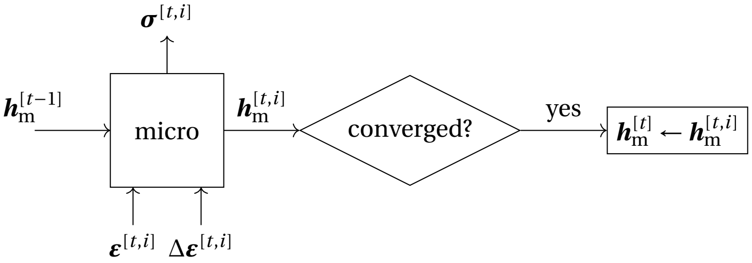

$\boldsymbol{\sigma}^{[t,i]}$
$\boldsymbol{h}_{\mathrm{m}}^{[t-1]}$
micro
$\boldsymbol{h}_{\mathrm{m}}^{[t,i]}$
converged?
yes
$\boldsymbol{h}_{\mathrm{m}}^{[t]} \leftarrow \boldsymbol{h}_{\mathrm{m}}^{[t,i]}$
$\boldsymbol{\varepsilon}^{[t,i]}$
$\Delta\boldsymbol{\varepsilon}^{[t,i]}$

Figure 3.5: Stress update procedure at each integration point in a macro model. The history parameters set $\boldsymbol{h}_{\mathrm{m}}$ contains accumulated plastic strain κ and stress tensor $\boldsymbol{\sigma}$ of each integration point of the micro model.

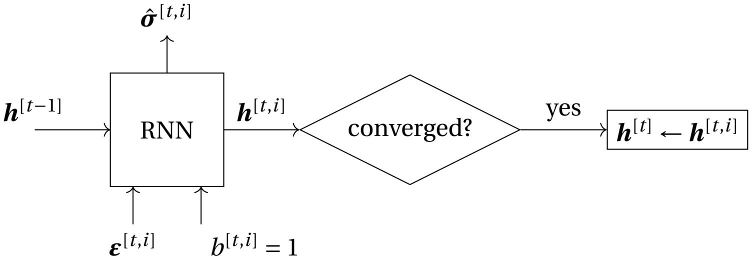

$\hat{\boldsymbol{\sigma}}^{[t,i]}$
$\boldsymbol{h}^{[t-1]}$
RNN
$\boldsymbol{h}^{[t,i]}$
converged?
yes
$\boldsymbol{h}^{[t]} \leftarrow \boldsymbol{h}^{[t,i]}$
$\boldsymbol{\varepsilon}^{[t,i]}$
$b^{[t,i]} = 1$

Figure 3.6: Stress update procedure at each integration point in a macro model when using the RNN model as a surrogate for the micro model. The history parameters are that of the RNN model which are defined in Section 3.3.2. We take the task of updating the history parameters to the macro model from the mLSTM cell by setting $b = 1$ and delegate it to the macro model.

macro strain to predict the stress. Then, the $\boldsymbol{h}_{\mathrm{m}}$ set of history parameters of the micro model, which consists of the accumulated plastic strain κ and the stress vector σ of each integration point of the micro model, is changed to the history vectors of the RNN model $\boldsymbol{h}$. Finally, since the control over updating history parameters is delegated to the macro model, we trivially set the input $b^{[t,i]}$ to one. With $b^{[t,i]}$ being one, the mLSTM cell always emits the updated value of history parameters.

From the implementation point of view, these changes can be incorporated into a parent multiscale code with minimal intrusion.

3.6.1. COMPUTING THE CONSISTENT TANGENT USING AUTOMATIC DIFFERENTIATION

Automatic differentiation is basically the application of chain rule to compute derivatives of numerical functions in a computer program [17]. Unlike numerical differentiation, it is exact up to the machine precision. And it differs from analytical differentiation by not requiring a closed form derivative expression.

Any deep learning library, such Tensorflow [18] and Pytorch [35], contains an automatic differentiation functionality that can be leveraged to compute the consistent tangent

$$\boldsymbol{D}^{[t,i]} = \frac{\partial \boldsymbol{\sigma}^{[t,i]}}{\partial \boldsymbol{\varepsilon}^{[t,i]}}. \tag{3.24}$$

Using a deep learning library's automatic differentiation functionality, the consistent tangent can be computed in exactly one line of code.

3.6.2. TECHNOLOGIES USED FOR IMPLEMENTATION

The entire framework is implemented in Python. In the implementation of the multiscale model we make use of the SciPy ecosystem [36]. We implement the RNN model using the Tensorflow library [18].

In the Python environment, interfacing Tensorflow library and SciPy ecosystem is readily possible. The output of the RNN model, the strain vector, in Tensorflow is a SciPy object, and therefore this output can immediately be used in the macro model, which is implemented using the SciPy ecosystem.

3.7. DISCUSSION ON COMPUTATIONAL COST

The computational cost of the RNN model is directly related to the size of the mLSTM cell k. The larger the value k, the larger the computational cost becomes.

Objectively comparing the computational cost of a standard FE-based micro model and that of an RNN model is very difficult. Nevertheless, based on the following remarks, we deduce that the RNN model is to be preferred:

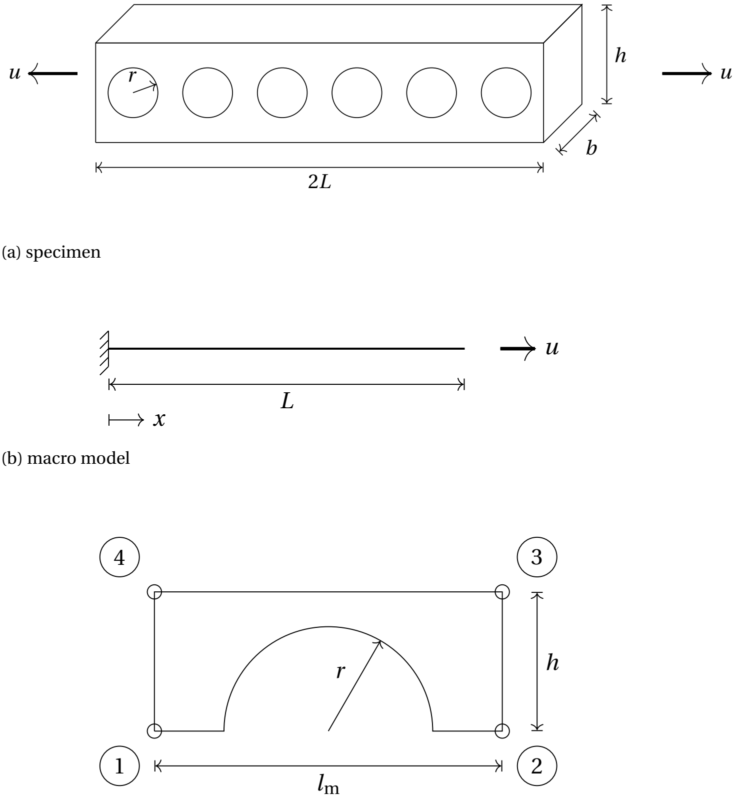
(1) the computational complexity of the RNN model is independent of the resolution of the micro model's mesh—one may liberally increase the FE mesh resolution and sample more accurate data, but train the same RNN model that was used for the data from the coarse mesh;

(2) the computational complexity of the RNN model is independent of that of the constitutive model of the micro model—one may enrich the constitutive model of the micro model, on the condition that patterns in the data do not completely change, and collect more accurate data, but train the same RNN model as the surrogate;

(3) among matrix operations, matrix inversion has the largest contribution to the computational cost in a micro model–luckily, there is no matrix inversion in an RNN model; and

(4) the evaluation of the constitutive model of the micro model also has a major contribution to the computational cost—the computational complexity of the RNN model is independent of that of the constitutive model and is also independent of the number of times the constitutive model has to be evaluated, i.e. the number of integration points in the micro model.

In Section 3.8, we cautiously report the computational speed up. At the same time, we acknowledge that the numbers that we report could vary significantly with different implementations.



(a) specimen

(b) macro model

(c) micro model

Figure 3.7: The length of the specimen (a) is assumed to be significantly larger than its height and width, therefore we model it as a one-dimensional macro model (b) and to incorporate the inclusions we consider the micro model (c).

3.8. Application

The specimen depicted in Figure 3.7a is a bar with a rectangular cross-section. The bar contains cylindrical inclusions along its length. We pull the bar by imposing a uniform displacement at its ends. The material parameters are: Young's modulus $E = 1000$ MPa; Poisson's ratio $\nu = 0.25$; yield stress $\sigma_\mathrm{Y} = 1$ MPa; viscosity parameter $\eta = 10^{-5}\ \mathrm{s}^{-1}$. Other model parameters are taken as $\beta = 1$, $a = -1$, and $b = 100$. The use of these parameters is discussed in Section 4.2.

We assume that the length of the bar is considerably larger than its width and height. Hence, we model it as a one-dimensional object. To incorporate the cylindrical inclusions we employ a multiscale scheme. We consider a one-dimensional homogeneous bar as the macro model and, due to symmetry, a two-dimensional plate with half a circle cut out of its bottom edge as the micro model. The macro and micro models are depicted in chapter-3/figures 3.7b and 3.7c, respectively.

At each integration point in the macro model, we impose the macro strain on the micro model as a set of Dirichlet boundary conditions. Since the macro strain is uniaxial, the micro model deforms uniaxially as well. To this end, we impose the following sets of boundary conditions on the micro model in Figure 3.7c:

$$\begin{aligned} \boldsymbol{u}_x^{\text{right}} &= \boldsymbol{u}_x^{\text{left}} + \boldsymbol{\varepsilon}\, l_{\text{m}} \\ \boldsymbol{u}_x^{\text{bottom}} &= \boldsymbol{0} \\ u_y^1 = u_x^1 = u_x^4 &= 0 \\ u_x^2 = u_x^3 &= \boldsymbol{\varepsilon}\, l_{\text{m}} \end{aligned} \tag{3.25}$$

Note that the micro model in Figure 3.7c does not statistically represent the heterogeneity of the object in Figure 3.7a. To truly construct a representative volume element (RVE), one needs to study the impact of considering more inclusions into the micro model. The performance of our RNN model however is independent from the statistical qualities of the micro model. We therefore employ the micro model in Figure 3.7c to prove the concept.

In the following sections we reduce the computational cost of the multiscale model by replacing the micro model with an RNN model.

3.8.1. USING AN RNN MODEL AS A SURROGATE FOR THE MICRO MODEL

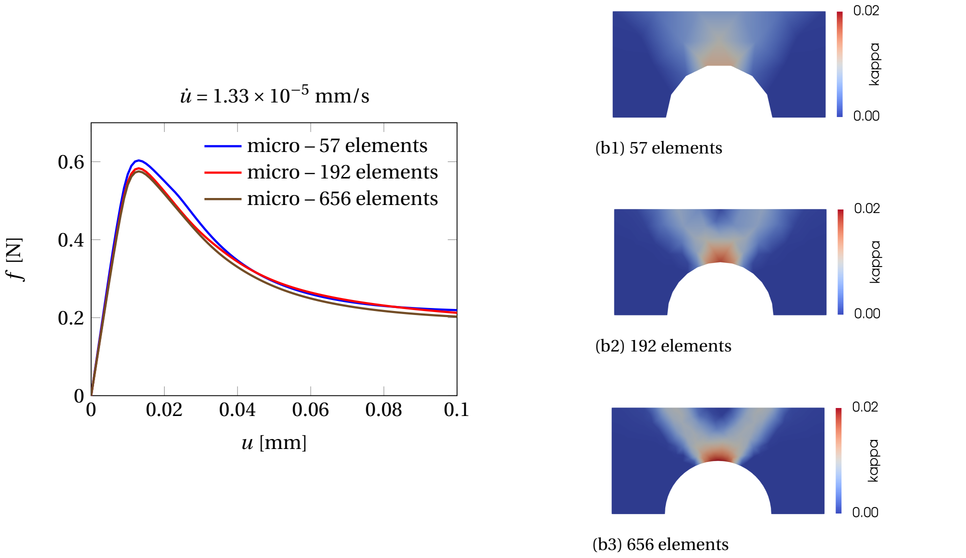
In this section we show that, in principle, the RNN model from Section 4.7 can be trained to serve as an efficient surrogate for the micro model.

We first conduct a mesh refinement study to identify a reasonable micro model discretization. For this example we consider the following measurements: length of the bar l = 10 mm; width of the bar b = 0.8 mm; height of the bar h = 1 mm; radius of inclusion r = 0.5 mm; length of the micro model l_m = 2 mm. We pull one end of the bar at a displacement rate $\dot{u}$ of 1.33×10^{-5} mm/s until it is pulled 2 mm. We clamp the other end.

We discretize the macro model using five one-dimensional linear elements. For the micro model, we employ three finite element models with increasing number of elements. In Figure 3.8a the reaction force at the left end of the macro model is plotted against the imposed displacement at its right end. Given the closeness of the results, we select the micro model with 192 linear triangular elements as the converged solution. To illustrate the failure mode of the specimen, we plot the accumulated plastic strain of all micro models in Figure 3.8b.

With the micro model chosen, as a sanity check, we reproduce the results of the micro model with the RNN model. To this end, we sample data using the strategy set forth in Section 3.3.1. In each integration point, at each time step and Newton-Raphson iteration, we collect the strain and the corresponding stress tensors. In this manner, as discussed in Section 3.3.1, we collect samples from both converged and not converged Newton-Raphson iterations. We then train an RNN model on these data. Through procedures explained in Section 3.5, we consider the size of the mLSTM cell $k = 200$ and the dropout probability $p = 0.5$.

We inject the trained RNN model in place of the micro model in the multiscale scheme. The force-displacement curves of the macro model with RNN are compared against that obtained with the micro model in Figure 3.9. The RNN is not only accurate, but is also considerably faster than the micro model. Based on our implementation, we gain two



(b1) 57 elements

(b2) 192 elements

(b3) 656 elements

(a) Reaction force f at left boundary of the macro model in Figure 3.7b against the imposed displacement u at the right boundary.

(b) Accumulated plastic strain at the end of simulations: (b1) blue curve, (b2) red curve, and (b3) black curve in Figure 3.8a.

Figure 3.8: A uniform mesh refinement study for the micro model. The micro model with 192 triangular elements is selected for the study.

orders of magnitude speed up. Nevertheless, we acknowledge that this speed up could significantly vary with a different implementation or a different micro model.

In this example we trained the RNN model on a set of stress-strain data collected from the macro model. We then used the RNN model to predict exactly the same data. This is a trivial exercise which serves as a sanity check. In practice we are interested in an RNN model that can predict and not merely reproduce the data it was trained on. In the next section we show that our RNN is capable of predicting data it did not previously observe.

3.8.2. THE RNN MODEL FOR A RANGE OF PARAMETERS

Let us consider the macro model is Figure 3.7 again. We increase the imposed displacement u linearly until it reaches a maximum value of u_{max} and then reverse the impose displacement until the imposed displacement goes back to zero. For this example we consider the following: length of the bar l = 100 mm; width of the bar b = 0.8 mm; height of the bar h = 1 mm; radius of inclusion r = 0.5 mm; length of the micro model l_m = 2 mm. The macro model is discretized with 50 linear elements and the micro model with 192 linear triangular elements.

The goal is to build a multiscale scheme that runs fast and is accurate in the following range of parameters: $u_{\text{max}} \in [0.5, 1.0]$ mm, $\dot{u} \in [1.33, 6.67] \times 10^{-4}$ mm/s. We follow the sampling strategy explained in Section 3.3.1. For the sake of simplicity, we uniformly sampled the parameters u_{max} and $\dot{u}$ in the aforementioned ranges. Specifically, we sam-

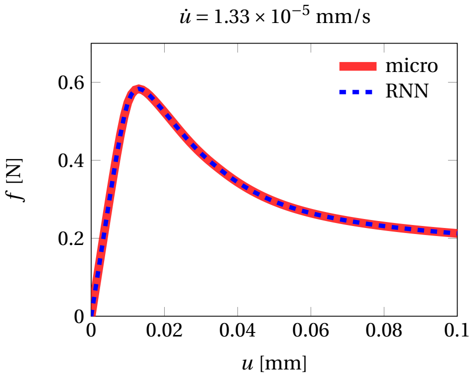


Figure 3.9: A sanity check: we train an RNN model on data generated from one configuration of macro model and plot the reaction force f at left boundary of the macro model in Figure 3.7b against the imposed displacement u at the right boundary.

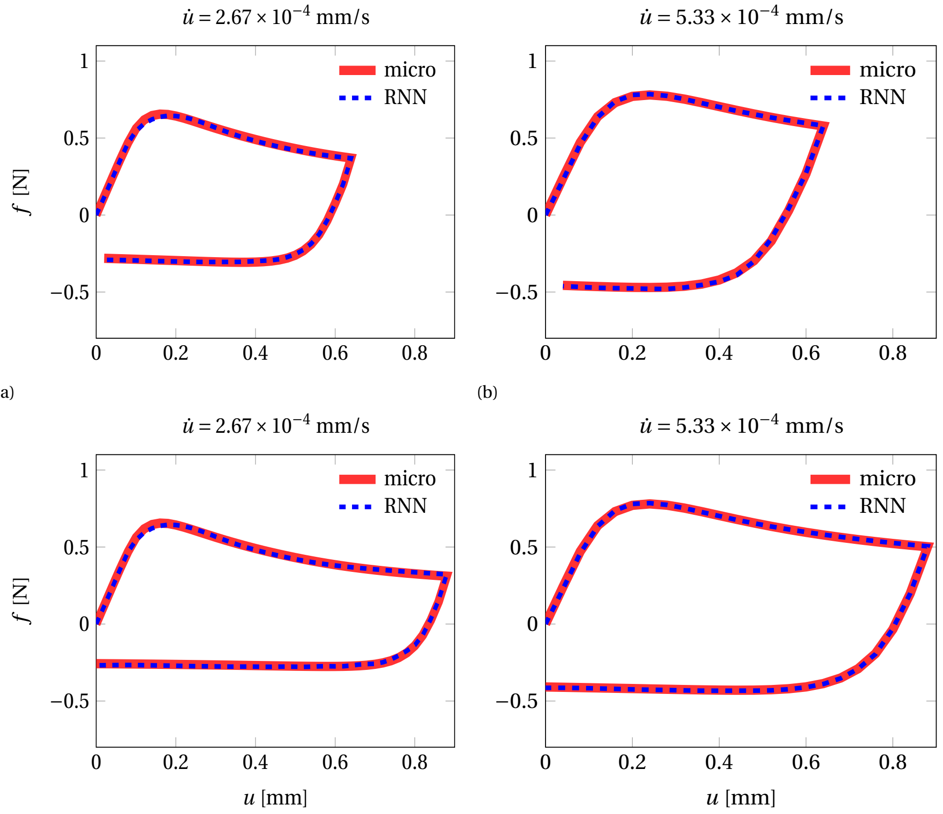
ple stress-strain data from the macro model with all possible permutations of the following set of parameters: $u_{\max} \in \{0.5, 0.75, 1.0\}$ mm, $\dot{u} \in \{1.33, 4, 6.67\} \times 10^{-4}$ mm/s. We then train the RNN model in Section 4.7 with the following set of hyperparameters: the size of the mLSTM cell $k = 200$, and the dropout rate $p = 0.5$. Again, these hyperparameters are chosen through a trial-and-error procedure as explained in Section 3.5. To test the performance of the RNN model, Figure 3.10 shows the force-displacement curves of the macro model with four sets of parameters $\{u_{\max}, \dot{u}\}$ that were not observed during training. Figure 3.10 shows that the RNN model performs accurately as long as the parameters $u_{\max}$ and $\dot{u}$ are in the range of data that the RNN model is trained on. The speed up for our implementation is three orders of magnitude. However, as soon as the parameters leave this range, the RNN model loses its accuracy. For instance in chapter-3/figures 3.11, the value of $u_{\max}$ is outside of this range. Basically, an RNN model learns the underlying patterns of the training samples. We collect training samples from a certain part of the parameter space related to $u_{\max}$ and $\dot{u}$. Consequently, the RNN model only learns patterns which are specific to that part of the parameter space. As a result, extrapolation is often inaccurate. Interestingly, in this specific case, the RNN model can still follow the trend.

In such scenarios, where we are interested in increasing the range of parameters that the RNN can perform accurately on, the RNN model should be retrained. We do so in the next section.

3.8.3. RETRAINING THE RECURRENT NEURAL NETWORK WITH NEW TRAIN DATA

In the previous application we showed that the RNN model is accurate when interpolating between training data but fails when extrapolating.

The straightforward solution to boost the accuracy of the RNN model is to retrain it. To this end, we collect new data in proximity of the parameters that the model performs poorly at, i.e. $\dot{u} = 10.67 \times 10^{-4}$ mm/s. Specifically, we extend the $u_{\max}$ set to $\dot{u} \in$



(a)

(b)

(c)

(d)

Figure 3.10: A comparison between accuracy of the micro model and that of the RNN model: we plot the reaction force f at left boundary of the macro model in Figure 3.7b against the imposed displacement u at the right boundary. None of these four cases was used in the training process.

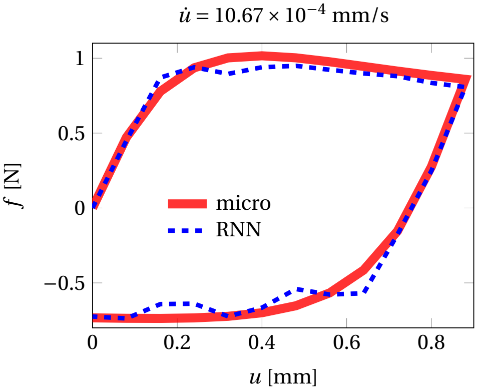


Figure 3.11: The RNN model loses accuracy when the imposed displacement rate $\dot{u}$ is out of the range of data the RNN model is trained on, i.e. $\dot{u} \notin [1.33, 6.67] \times 10^{-4}$ mm/s. We show this by plotting the reaction force f at left boundary of the macro model in Figure 3.7b against the imposed displacement u at the right boundary

$\{1.33, 4, 6.67, 13.33\} \times 10^{-4}$ mm/s. We intentionally make sure that $\dot{u} = 10.67 \times 10^{-4}$ mm/s is not included during the training process to illustrate the predictive capability of the RNN model. Unlike the procedure discussed in Section 4.5.3, where we initialized the trainable parameters to random numbers, here we initialize them to their previously optimized values.

To show the enhanced performance of the retrained model, we plot the force-displacement curves of the same cases as in previous example. chapter-3/figures 3.13 show that the RNN-based multiscale model still performs accurately in all previous scenarios. And thanks to retraining, it is also accurate when $\dot{u} = 10.67 \times 10^{-4}$ mm/s as shown in Figure 3.12.

In practice, it has been observed that the performance, in terms of accuracy, of a neural network increases by enriching the data set and the size of the neural network [37]. Objectively comparing the impact of the size of the train set on the performance of a neural network is however not a trivial task. One of the rare instances of such study can be found in Reference [38].

Until this point the macro model exhibited a constant strain field. Therefore, the prediction of the RNN model was the same at all integration points in the macro model. In the next section we consider a FEM model in which the strain field localizes.

3.8.4. STRAIN LOCALIZATION IN A ONE-DIMENSIONAL BAR

To simulate strain localization, we introduce an imperfection at the center of the bar in Figure 3.14. The width of the imperfection zone is 0.9 times the area of the rest of the bar and its length is one tenth of the length of the bar. The other measurements are the same as in the previous example. We pull one end of the bar at the rate of $\dot{u}$ until it is pulled by 2 mm; the other end is clamped. The bar is discretized with 50 linear finite elements elements, five of which are in the weak zone.

We acknowledge that the response of the macro model, due to strain softening and localization, is mesh dependent. Removing such pathological mesh dependence in a

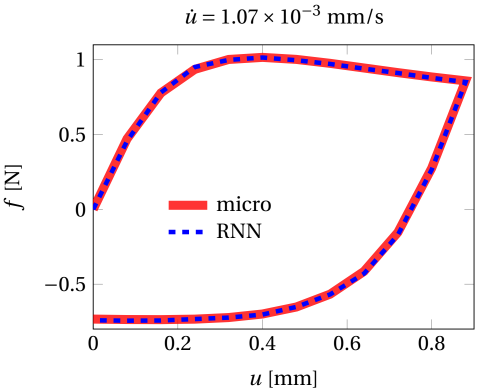
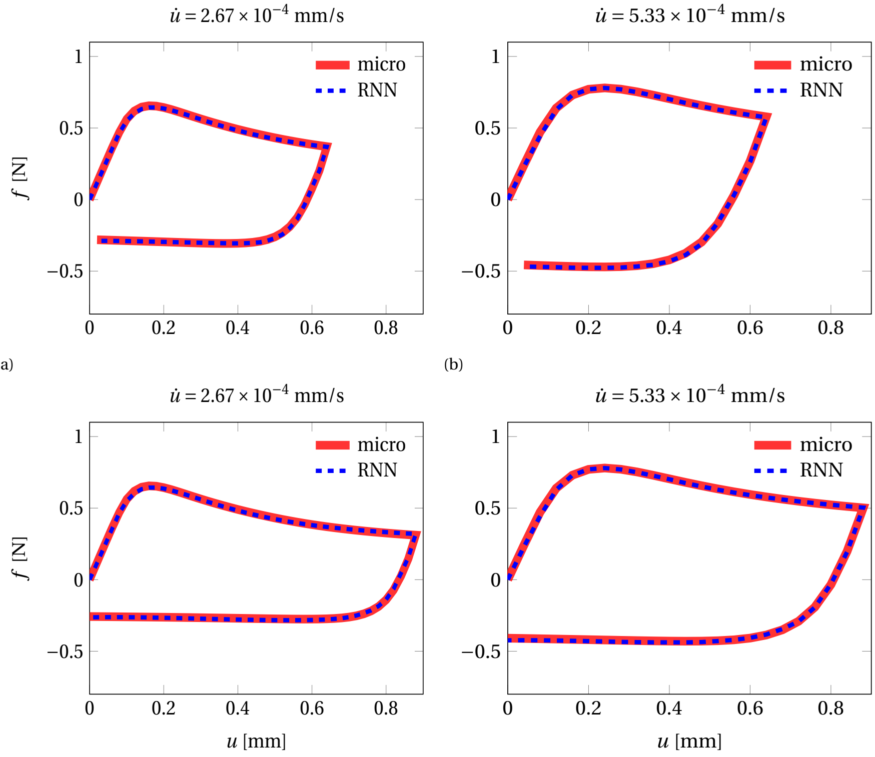


Figure 3.12: Comparing this figure with Figure 3.11, it is clear that considering an enriched set of data, i.e. $\dot{u} \in \{1.33, 4, 6.67, 13.33\} \times 10^{-4}$ mm/s, boosts the accuracy of the RNN model.



(a)

(b)

(c)

(d)

Figure 3.13: The RNN model trained on the enriched set of data performs as accurately as it used to do in Figure 3.10.

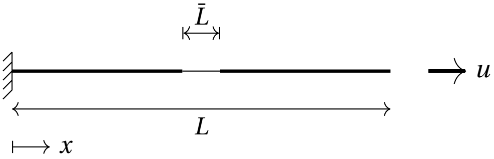


Figure 3.14: An imperfection of length $\bar{L} = 0.9L$ is introduced at the center of the macro model in Figure 3.7b.

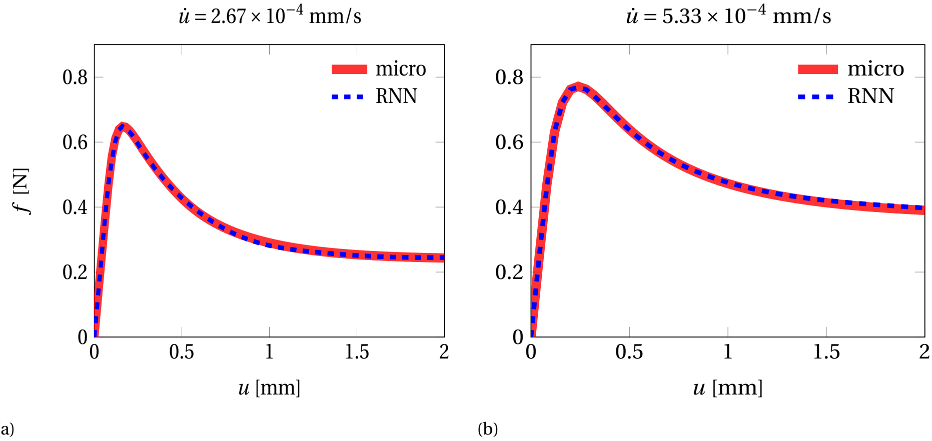


Figure 3.15: The reaction force f at left boundary of the macro model in Figure 3.7b against the imposed displacement u at the right boundary. The average response of the macro model in Figure 3.14 appears to be accurate when using the RNN in place of the micro model.

multiscale method, to the extend of our knowledge, remains an open question if continuum models are employed at both scales. In this example our goal is to build a multiscale scheme that runs fast and is accurate for the previously specified parameters and all values of the imposed displacement rate $\dot{u}$ in the range $[1.33, 6.67] \times 10^{-4}$ mm/s.

Following the strategy set forth in Section 3.3.1, we collect data from the macro model with $\dot{u} \in \{1.33, 4, 6.67\} \times 10^{-4}$ mm/s. Then we train the RNN model with size of the mLSTM cell $k = 200$ and rate of dropout $p = 0.5$. Coincidentally, this RNN model is identical to that of the preivous example. We reached these hyperparameters through hyperparameter tuning strategies mentioned in Section 3.5.

We test the accuracy of the model by testing it with imposed displacement rates that were not observed during training, $\dot{u} \in \{2.67, 5.33\} \times 10^{-4}$ mm/s. The force-displacement curves are depicted in Figure 3.15. It appears that the RNN model performs accurately as the micro model surrogate in terms of structural response of the macro model. We further zoom into strain fields and stress-strain response of the macro model to study its response at the element level.

The strain fields in the macro model, depicted in chapter-3/figures 3.16, show strain localization at the center of the bar. We observe that the strain field localizes more when $\dot{u} = 2.67 \times 10^{-4}$ mm/s (Figure 3.16a) than when $\dot{u} = 5.33 \times 10^{-4}$ mm/s (Figure 3.16b). We also observe that as the strain field localizes further, the discrepancy between the

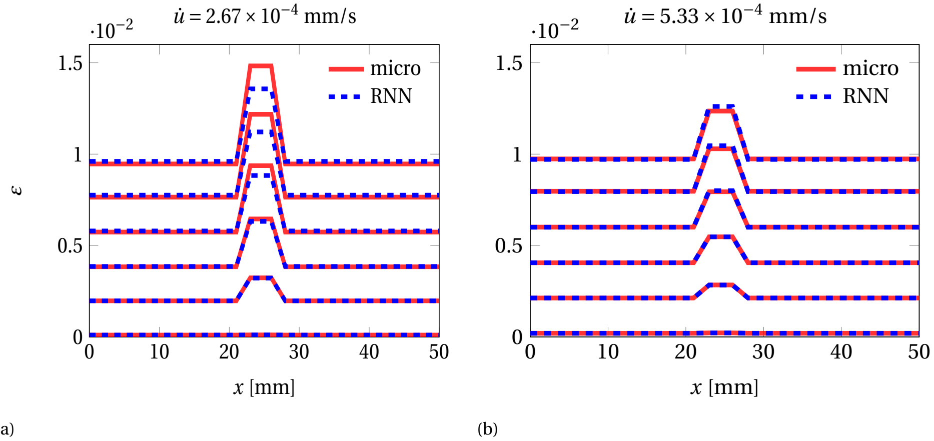


Figure 3.16: Evolution of the axial strain field. From Figure 3.16a it is apparent that the accuracy of predicted strain field diminishes when the rate of imposed displacement $\dot{u} = 2.67 \times 10^{-4}$ mm/s. In Section 3.8.4 we remedy this issue by retraining the RNN model on an enriched training set.

RNN-based multiscale model and the classical one becomes greater. This discrepancy can be resolved by retraining the RNN model in fashion similar to that discussed in Section 3.8.3. We do so in the next section.

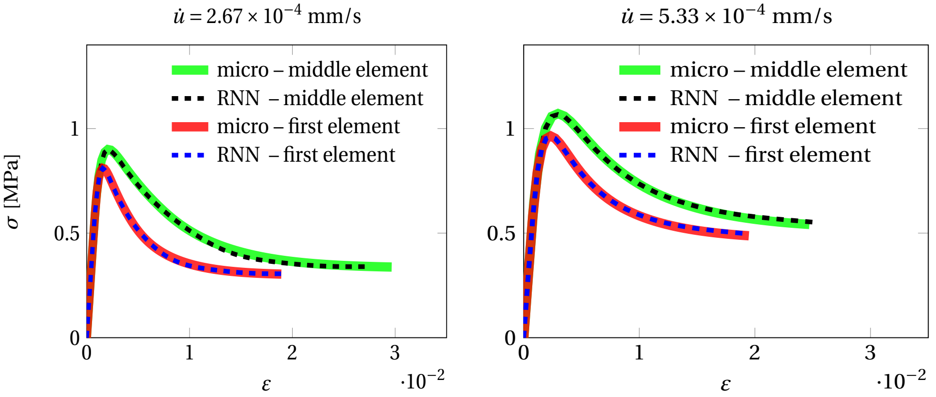
Let us now study the stress-strain curves, chapter-3/figures 3.17a and 3.17b, of an element inside the localization region, and another one outside it. We observe that when the strain rate is smaller and strain localization is greater, the discrepancy between RNN and micro model predictions become larger. In the next section we show that by retraining the RNN model on an enriched set of samples this discrepancy disappears.

3.8.5. Strain localization in a one-dimensional bar: Retraining

In the previous example we observed that the RNN model is less accurate when the strain rate is smaller. The straightforward way to increase the accuracy is to retrain the RNN model with an enriched set of samples. Specifically, we expand the data set by collecting data from the macro model with strain rates $\dot{u} \in \{1.33, 2, 3.33, 4, 6.67\} \times 10^{-4}$ mm/s. Note that we deliberately kept $\dot{u} \in \{2.67, 5.33\} \times 10^{-4}$ mm/s out of the data set that we use for training to assess the predictive capability of the RNN model.

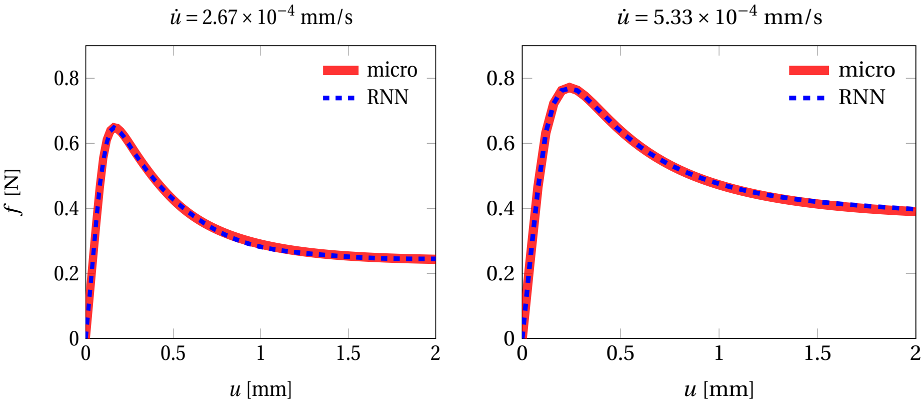
From Figure 3.18, we observe that the force-displacement curves are still accurate. We observe a significant boost in the accuracy of the strain field when the strain rate is 2.67×10^{-4} mm/s by comparing Figure 3.19a and 3.16a. The accuracy of the stress-strain curve in Figure 3.20a is slightly improved when comparing it with that of in Figure 3.17a.

Let us consider a classical constitutive model to create an analogy for retraining. One tunes a constitutive model based on a set of experimental samples. The more experimental samples one acquires, the more accurate the constitutive model becomes. The computational complexity of evaluating a constitutive model is however independent of the amount of experimental samples. A similar procedure is in order here. The more samples we train the RNN model on, the more accurate the RNN model becomes. The



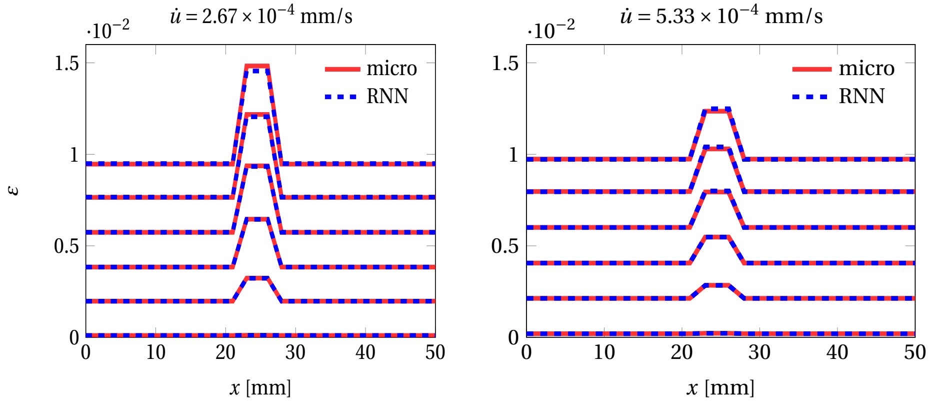
(a) (b)

Figure 3.17: Strain-strain response of an element in the middle of the macro model, depicted in Figure 3.14, and the element adjacent to the clamped boundary.



(a) (b)

Figure 3.18: Reaction force f at left boundary of the macro model in Figure 3.14 against the imposed displacement u at the right boundary. Comparing this figure with Figure 3.15, it appears that the average response of the system remains accurate when the RNN model is trained on an enriched set of data.



(a)

(b)

Figure 3.19: Evolution of the axial strain field. Comparing this figure with Figure 3.16 it is clear that, by training the RNN model on an enriched set of data, the accuracy of the predicted strain field increases.

computational cost of training the RNN model inevitably becomes larger. The computational complexity of evaluating it however remains the same.

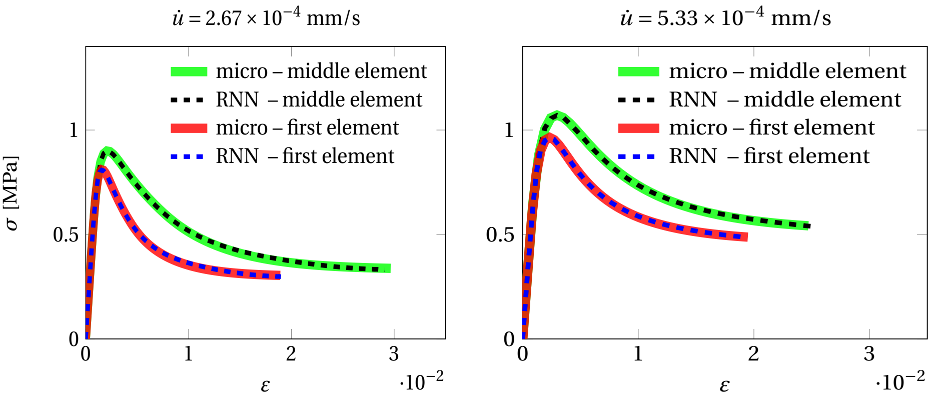
Up until this point we trained two RNN models. One which only works for the macro model with no imperfection and the other for the macro model which exhibits strain localization. In the next section we train one RNN model that works for both these macro models at the same time.

3.9. A RECURRENT NEURAL NETWORK THAT WORKS FOR A SET OF MACRO MODELS

The samples that we collected from the two macro models in the previous sections are generated from an identical constitutive model, namely the micro model with Perzyna viscoplasticity. The RNN model is essentially a surrogate for this constitutive model. Therefore, it is reasonable to expect that one RNN model can be trained on both sets of data simultaneously. In this section we put this hypothesis to a test. Specifically, we collect the samples from Section 3.8.3 and 3.8.5 in one data set. We train the RNN model with the same hyperparameters as in previous examples on this set.

We execute the simulations from chapter-3/figures 3.13a, 3.13b, 3.13c, 3.13d, 3.12, 3.15a, and 3.15b with the newly trained RNN model. We depict the result of these simulations in chapter-3/figures 3.21a, 3.21b, 3.21c, 3.21d, 3.21e, 3.22a, and 3.22b, respectively. It is evident that the RNN model, which is trained on the data from both macro models in the previous sections, performs accurately.

The sampling strategy put forth in Section 3.3.1, on the one hand, sacrifices the versatility of the RNN model. For instance, in this example, the RNN model can only reliably be used in the macro models depicted in chapter-3/figures 3.7b and 3.14. On the other hand however, it dramatically simplifies the previously very difficult task of sampling. And hence, it makes the application of the RNN model viable.



(a)

(b)

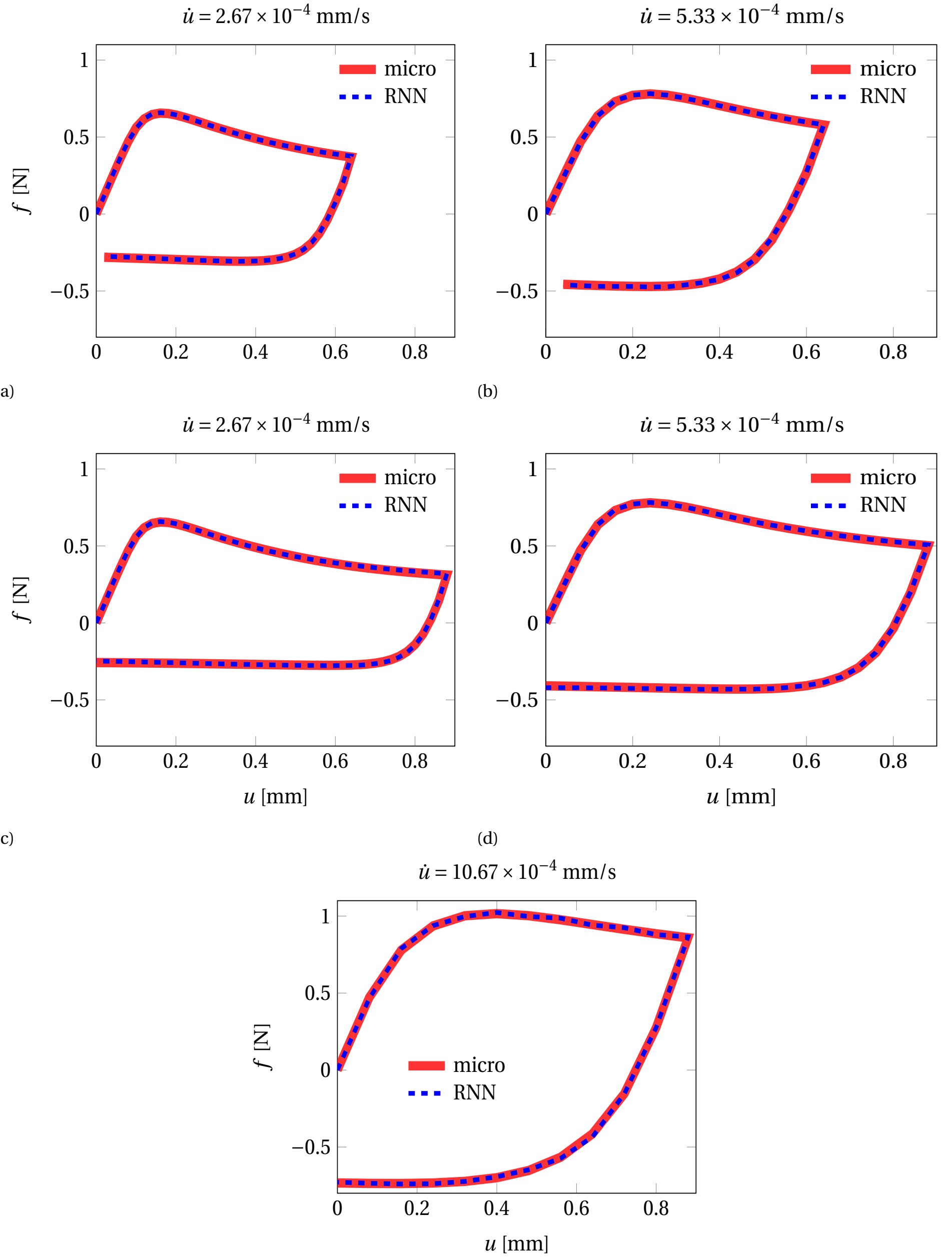
Figure 3.20: Stress-strain response of an element in the middle of the macro model depicted in Figure 3.14 and the element adjacent to the clamped boundary. Comparing Figure 3.17a with Figure 3.20a, it appears that the accuracy of the stress-strain curve of the middle element (dashed black curve) is slightly increased. This is due to the fact that the RNN model is trained on an enriched set of data.

3.10. Conclusions

In this contribution we introduced an RNN model that serves as an efficient surrogate for the micro model in a multiscale finite element analysis. We trained the RNN model on a set of data generated by a micro model. The data in the set are collected using an efficient sampling technique that we proposed. We also discussed how to implement the RNN model in a multiscale scheme and how to compute the consistent tangent using automatic differentiation. Finally, we assess the performance of the RNN model through a series of academic tests, and we observe that its accuracy can be boosted through retraining on an enriched data set.

In our experiments, the RNN model is remarkably faster than the FE-based micro model (note that this is not a universal fact as one can easily construct an RNN model that is computationally more expensive than a finite element model). Despite the runtime speed improvements compared to classical FEM simulations, the training time of the RNN model is considerable. A reasonable question is whether this efficient RNN model is worth the time spent on its training. To answer this question one needs to consider the application at hand. Assume for instance that the runtime of a multiscale model is 10 hours, and it takes 1000 hours to train an RNN model. The runtime of the RNN-enhanced multiscale model is only 1 hour. Then, the tenfold speed up becomes only useful if we are interested in running more than 112 simulations (runtime of the multiscale model for 112 simulations: $112 \times 10 = 1,120$ hours; runtime of the RNN-enhanced multiscale model $1,000 + 112 \times 1 = 1,112$ hours).

In our experiments, the RNN model appears to be very accurate (it is to be noted however that there exist no guarantee that this accuracy can always be achieved) and, if needed, we have the means (by retraining) to boost its accuracy. However, the parameters of the RNN model are non-physical. The question that arises here is whether a non-



(a)

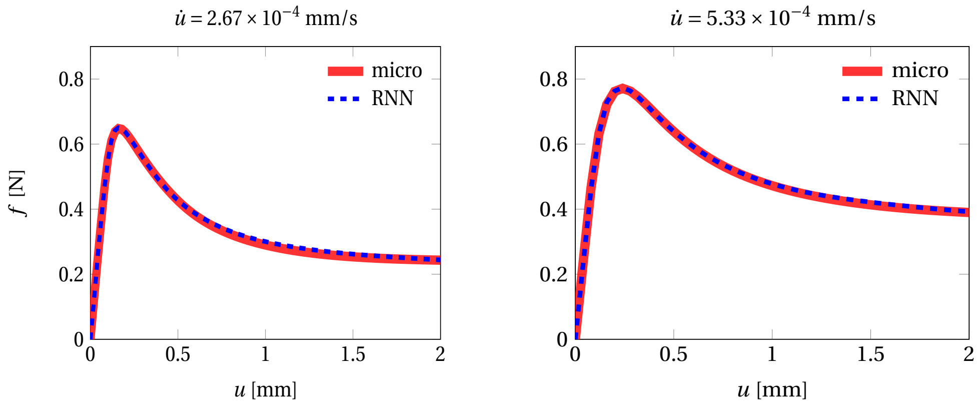
(b)

(c)

(d)

(e)

Figure 3.21: Reaction force f at left boundary of the macro model in Figure 3.7b against the imposed displacement u at the right boundary. The RNN model still performs very accurately when trained on a set of data from both macro models shown in chapter-3/figures 3.7b and 3.14.



(a)

(b)

Figure 3.22: Reaction force f at left boundary of the macro model in Figure 3.14 against the imposed displacement u at the right boundary. The RNN model still performs very accurately when trained on a set of data from both macro models show in chapter-3/figures 3.7b and 3.14.

physical and completely data-driven model is reliable. The RNN model that is developed in this work is physically-reliable as long as it is accurate. The RNN model is essentially a surrogate for a physics-based micro model. As a result, this model approximates the response of the physics-based micro model. Hence, it approximately satisfies its physical features. The more accurate the RNN model, the more accurately the physical features of the micro model are reproduced. In any case, it is also possible to measure the degree of reliability of the RNN prediction by developing a prediction interval [39] (a prediction interval is an estimate of an interval in which the predictions of the RNN model will fall, with a certain probability).

A trained RNN model is at best capable of interpolating between the training data. So, if the RNN model is trained on data from an uniaxial loading condition, it cannot accurately predict bending. However, one may train the RNN model on data from both uniaxial and bending conditions. The RNN model trained in this way will be accurate for both uniaxial and bending loading conditions.

An interesting characteristic of an RNN model is that by training it on more data (for instance, data from different loading conditions) and at the same time increasing the size of the RNN (either size of each layer, or number of layers) its accuracy increases. This characteristic suggests a fundamental trade-off between range of patterns the model can learn and its computationally complexity. The more variety of loading conditions the RNN model is trained on, the larger the RNN needs to be to maintain an acceptable level of accuracy. However, the larger the RNN, the more computationally expensive it becomes. Ultimately, it is possible to construct an RNN model that, in spite of being accurate, is much more computationally demanding than a corresponding FEM model.

Can we construct an RNN model on a set of data collected from lab experiments? In our opinion, such experimentally-trained RNN model would not perform well. One reason is that the trained RNN model can only map a set of observable variables to another observable variable and it neglects other dependencies. For instance, if the experiments measure uniaxial stress against the applied uniaxial strain, the RNN model only learns how to map the uniaxial strain to the uniaxial stress, and it completely ignores the dependency of the stress on other strain components. This implies that one would need a very large experimental data set, with results related to all possible stress-strain combinations. The other reason is that since there exist noise in experimental data, the trained RNN model would most probably learn some patterns in the noise and therefore fail to satisfy some physical laws. This is in contrast with what has been done in this contribution where the data collected from a physical model are noiseless. As a consequence of failing to satisfy some physical laws, even if the model performs accurately, serious concerns arise regarding its reliability and generality.

REFERENCES

[1] F. Ghavamian and A. Simone, *Accelerating multiscale finite element simulations of history-dependent materials using a recurrent neural network,* Computer Methods in Applied Mechanics and Engineering **357**, 112594 (2019).

[2] F. Feyel, *A multilevel finite element method (FE^2) to describe the response of highly non-linear structures using generalized continua,* Computer Methods in Applied Mechanics and Engineering **28–30**, 3233 (2003).

[3] L. C. Jain and L. R. Medsker, *Recurrent Neural Networks: Design and Applications*, 1st ed. (CRC Press, Inc., Boca Raton, FL, USA, 1999).

[4] I. Sutskever, *Training Recurrent Neural Networks*, Doctoral dissertation, University of Toronto (2013).

[5] Z. C. Lipton, J. Berkowitz, and C. Elkan, *A critical review of recurrent neural networks for sequence learning,* arXiv e-prints , arXiv:1506.00019 (2015), arXiv:1506.00019 [cs.LG] .

[6] R. Hambli, H. Katerchi, and C. L. Benhamou, *Multiscale methodology for bone remodelling simulation using coupled finite element and neural network computation,* Biomechanics and Modeling in Mechanobiology **10**, 133 (2011).

[7] J. L. Chaboche, *Continuous damage mechanics – A tool to describe phenomena before crack initiation,* Nuclear Engineering and Design **64**, 233 (1981).

[8] X. Lu, D. G. Giovanis, J. Yvonnet, V. Papadopoulos, F. Detrez, and J. Bai, *A data-driven computational homogenization method based on neural networks for the nonlinear anisotropic electrical response of graphene/polymer nanocomposites,* Computational Mechanics **64**, 307 (2019).

[9] X. Lu, J. Yvonnet, F. Detrez, and J. Bai, *Multiscale modeling of nonlinear electric conductivity in graphene-reinforced nanocomposites taking into account tunnelling effect,* Journal of Computer Physics **337**, 116 (2017).

[10] L. Xia and P. Breitkopf, *A reduced multiscale model for nonlinear structural topology optimization,* Computer Methods in Applied Mechanics and Engineering **280**, 117 (2014).

[11] F. Fritzen and M. Hodapp, *The finite element square reduced (FE^{2R}) method with GPU acceleration: towards three-dimensional two-scale simulations,* International Journal for Numerical Methods in Engineering **107**, 853 (2016).

[12] M. Caicedo, J. L. Mroginski, S. Toro, M. Raschi, A. Huespe, and J. Oliver, *High performance reduced order modeling techniques based on optimal energy quadrature: Application to geometrically non-linear multiscale inelastic material modeling,* Archives of Computational Methods in Engineering (2018), 10.1007/s11831-018-9258-3.

[13] I. B. C. M. Rocha, F. P. van der Meer, S. Raijmaekers, F. Lahuerta, R. P. L. Nijssen, L. P. Mikkelsen, and L. J. Sluys, *A combined experimental/numerical investigation on hygrothermal aging of fiber-reinforced composites,* European Journal of Mechanics - A/Solids **73**, 407 (2019).

[14] K. Wang and S. WaiChing, *A multiscale multi-permeability poroplasticity model linked by recursive homogenizations and deep learning,* Computer Methods in Applied Mechanics and Engineering **334**, 337 (2018).

[15] M. A. Bessa, R. Bostanabad, Z. Liu, A. Hu, D. W. Apley, C. Brinson, W. Chen, and W. K. Liu, *A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality,* Computer Methods in Applied Mechanics and Engineering **320**, 633 (2017).

[16] A. Karpathy, *The unreasonable effectiveness of recurrent neural networks,* (2015), [Online; accessed January 2019].

[17] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, *Automatic differentiation in machine learning: A survey,* Journal of Machine Learning Research **18**, 1 (2018).

[18] 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.

[19] F. Ghavamian, P. Tiso, and A. Simone, *POD-DEIM model order reduction for strain-softening viscoplasticity,* Computer Methods in Applied Mechanics and Engineering **317**, 458 (2017).

[20] D. Peric, E. A. de Souza Neto, R. A. Feijoo, M. Partovi, and A. J. Carneiro Molina, *On micro-to-macro transitions for multi-scale analysis of non-linear heterogeneous materials: unified variational basis and finite element implementation,* International Journal for Numerical Methods in Engineering **87**, 149 (2010).

[21] V. P. Nguyen, O. Lloberas-Valls, M. Stroeven, and L. J. Sluys, *Computational homogenization for multiscale crack modeling. Implementational and computational aspects,* Journal of Computational and Applied Mathematics **234**, 2175 (2010).

[22] R. Everson and L. Sirovich, *Karhunen-Loève procedure for gappy data,* Journal of the Optical Society of America A **12**, 1657 (1995).

[23] K. Carlberg, C. Bou-Mosleh, and C. Farhat, *Efficient non-linear model reduction via a least-squares Petrov-Galerkin projection and compressive tensor approximations,* International Journal for Numerical Methods in Engineering **86**, 155 (2011).

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: A simple way to prevent neural networks from overfitting,* Journal of Machine Learning Research **15**, 1929 (2014).

[25] S. Hochreiter, *Untersuchungen zu dynamischen neuronalen Netzen,* Diploma thesis, Technical University of Munich (1991).

[26] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, *Gradient flow in recurrent nets: The difficulty of learning long-term dependencies,* in *A Field Guide to Dynamical Recurrent Neural Networks*, edited by S. C. Kremer and J. F. Kolen (IEEE press, New York, 2001) Chap. 14, pp. 237–244.

[27] S. Hochreiter and J. Schmidhuber, *Long short-term memory,* Neural Computation **9**, 1735 (1997).

[28] C. Olah, *Understanding LSTM Networks,* (2015), [Online; accessed January 2019].

[29] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches,* arXiv e-prints , arXiv:1409.1259 (2014), arXiv:1409.1259 [cs.CL] .

[30] L. Fei-Fei, J. Johnson, and S. Yeung, *CS231n Convolutional Neural Networks for Visual Recognition,* (2018), [Online; accessed January 2019].

[31] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization,* in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, edited by Y. Bengio and Y. LeCun (2015).

[32] G. Goh, *Why momentum really works,* (2017), [Online; accessed January 2019].

[33] J. Snoek, H. Larochelle, and R. P. Adams, *Practical Bayesian optimization of machine learning algorithms,* in *Advances in Neural Information Processing Systems 25, IEEE Conference on Neural Information Processing Systems–Natural and Synthetic* (Massachusetts Institute of Technology Press, 2012) pp. 2951–2959.

[34] D. Geng and S. Shih, *Machine learning crash course: Part 4 - The bias-variance dilemma,* (2017), [Online; accessed January 2019].

[35] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, *Automatic differentiation in pytorch,* in *NIPS 2017 Autodiff Workshop* (2017).

[36] E. Jones, T. Oliphant, P. Peterson, *et al.*, *SciPy: Open source scientific tools for Python,* (2001–), [Online; accessed January 2019].

[37] A. Ng, *Nuts and bolts of applying deep learning,* (2016), [Online; accessed January 2019].

[38] D. Soekhoe, P. van der Putten, and A. Plaat, *On the impact of data set size in transfer learning using deep neural networks,* in *Advances in Intelligent Data Analysis XV*, edited by H. Boström, A. Knobbe, C. Soares, and P. Papapetrou (Springer International Publishing, 2016) pp. 50–60.

[39] A. Khosravi, S. Nahavandi, D. Creighton, and A. F. Atiya, *Comprehensive review of neural network-based prediction intervals and new advances,* IEEE Transactions on Neural Networks **22**, 1341 (2011).

4

A CONVOLUTION NEURAL NETWORK SURROGATE FOR FINITE ELEMENT ANALYSIS OF MULTI-PHYSICS PROBLEMS

Artificial intelligence is the new electricity.

Andrew Ng

We propose HydraNet, a convolutional neural network (CNN), as an efficient surrogate for finite element analysis of multi-physics problems. HydraNet is considerably more efficient than a FEM analysis because its evaluation does not require any matrix inversions or evaluation of complex constitutive models. In this contribution, we demonstrate the exceptional performance of HydraNet as a surrogate solver for the multi-physics FEM analysis of a lithium-ion battery. We also discuss the impact of hyperparameters on the accuracy of the surrogate.

4.1. INTRODUCTION

Multi-physics models enable a more realistic analysis of complex materials compared to traditional single-physics models. These models are essentially a set of coupled partial differential equations (PDE), each describing a different physics. The system of PDE is usually solved using a numerical method such as the finite element method (FEM) which, in most cases, is computationally too expensive to be used in multi-query applications (typical examples are multiscale analyses or shape optimization procedures). In this contribution, we tackle this issue by proposing a convolutional neural network (CNN) [1] variant as an efficient surrogate for multi-physics FEM analysis. The approach is demonstrated on an academic problem of an all-solid-state micro-battery cell.

The need to speed up battery simulations has long been recognized, and neural networks are among the most promising approaches [2–8]. Here we consider convolutional neural networks, which are a class of deep neural networks commonly used in the computer vision field. In the past decade these techniques have revolutionized the image understanding field and this success has not gone unnoticed in other research areas. In the field of computational science and engineering most of the contributions revolve around the replacement of computationally expensive numerical procedures with more efficient CNN-based models. We refer for instance to Cecen et al. [9] and Li et al. [10] who employ a CNN model to predict material properties of a specimen by processing its geometry, Tompson et al. [11] and Winovich et al. [12] who exploit a CNN model to replace the computational expensive matrix inversion in a FEM analysis, Jiang et al. [13] who use a CNN model to quantify the mechanical degradation (active NMC particles-carbon/binder domain detachment) of a charged battery electrode, Sasaki et al. [14] who use a CNN model to replace an entire multi-query pipeline, and finally Khan et al. [15] and Liang et al. [16] who propose CNN models that take the geometry and material properties of a specimen and predict a certain solution field. Our contribution falls into the same category as that of Khan et al. [15] and Liang et al. [16]. At variance with [15], we consider a multi-physics problem for which our CNN model takes the geometry of a battery cell and predicts all fields associated with the set of PDEs. And at variance with Liang et al. [16], we encode the geometry of the battery cell into a set of image-like fields as detailed in Section 4.4.2.

We consider the simulation of the battery cell behavior as an archetype of a multi-physics problem. In this work, the problem is fully characterized by means of the model employed by Grazioli et al. [17], briefly summarized in Section 4.2, who studied the effect of mechanical stresses arising in solid polymer electrolytes on the electrochemical performance of all-solid-state micro-batteries. The simulation results obtained with the FEM implementation of that model are used to develop a CNN-based surrogate for this type of FEM analysis. To the best of our knowledge, Hydranet is the first application of neural network to accelerate multiphysics (electrochemical-mechanical) simulations of battery cells.

A vanilla CNN model has one body and one head. The body takes in the input, here the geometry, and the head predicts a set of outputs, here understood as outputs of the FEM analysis. To use a vanilla CNN as a surrogate for multi-physics problems there exist two approaches: 1) employ one vanilla CNN model for each PDE or 2) employ one CNN model for all PDEs. In the former scenario, the body of each CNN model performs the same repeated task, thus rendering the surrogate model inefficient. In the latter scenario, if one PDE is more complex than another, there is no possibility to increase the capacity of the CNN model only for that specific PDE without unnecessarily increasing the capacity of the CNN model for others. To tackle these issues, we propose the use of HydraNet as discussed in Section 4.5. The peculiarity of HydraNet is its architecture: HydraNet has one body and several heads, and each head could be associated with a specific PDE allowing for its parameters to be tuned independently. Features and components of HydraNet are described in a progressive manner until the full model is applied as a surrogate for the coupled electrochemical-mechanical battery cell model employed by Grazioli et al. [17] in Section 4.7.

4.2. Problem statement in a FEM context

We consider the steady-state version of the electrochemical-mechanical model used by Grazioli et al. [17]. For brevity, we provide a summary of the formulation and refer the reader to Refs. [17, 18] for details. A representative slice of the all-solid-state micro-battery with trench geometry under investigation is shown in Fig. 4.1. Subscripts Pos, Neg and SPE identify quantities related to each cell component, i.e., positive and negative electrodes and solid polymer electrolyte, respectively. The model is formulated in terms of three primary field variables: electrostatic potential ϕ, lithium concentration c, and displacements $\boldsymbol{u}$. Each field variable depends on location $\boldsymbol{x} \in V_k$, where V_k identifies the domain of validity, i.e., k = Pos, Neg or SPE. The modeling of the electrodes is limited to the charge conservation equation for conductive materials and only involves field variable ϕ. A two-way coupling between electrochemistry and mechanics is assumed in the solid polymer electrolyte, whose model therefore involves field variables ϕ, c, and $\boldsymbol{u}$.

Charge conservation in the electrodes obeys the relationship

$$\nabla \cdot \boldsymbol{i}_{\mathrm{el}} = 0, \quad \boldsymbol{x} \in V_{\mathrm{el}}, \quad \mathrm{el} = \mathrm{Pos}, \mathrm{Neg}, \tag{4.1}$$

where $\boldsymbol{i}$ is the electric current density. The behavior of the SPE is described by the charge conservation equation

$$\nabla \cdot \boldsymbol{i}_{\mathrm{SPE}} = 0, \quad \boldsymbol{x} \in V_{\mathrm{SPE}}, \tag{4.2}$$

the mass conservation equation in a steady-state regime

$$\nabla \cdot \boldsymbol{h}_{\mathrm{SPE}} = 0, \quad \boldsymbol{x} \in V_{\mathrm{SPE}}, \tag{4.3}$$

and the equilibrium equation without body forces

$$\nabla \cdot \boldsymbol{\sigma}_{\mathrm{SPE}} = \boldsymbol{0}, \quad \boldsymbol{x} \in V_{\mathrm{SPE}}. \tag{4.4}$$

In the above equations, $\boldsymbol{h}$ and $\boldsymbol{\sigma}$ represent the mass flux vector and the stress tensor, respectively.

The relationships between $\boldsymbol{i}$, $\boldsymbol{h}$, and $\boldsymbol{\sigma}$, the primary field variables, and their gradients, are summarized by the following expressions:

$$\boldsymbol{i}_{\mathrm{el}} = f_{\boldsymbol{i}_{\mathrm{el}}}\left(\nabla\phi_{\mathrm{el}}\right), \quad \mathrm{el} = \mathrm{Pos}, \mathrm{Neg} \tag{4.5a}$$

$$\boldsymbol{i}_{\mathrm{SPE}} = f_{\boldsymbol{i}_{\mathrm{SPE}}}\left(c_{\mathrm{SPE}}, \nabla\phi_{\mathrm{SPE}}, \nabla c_{\mathrm{SPE}}, \nabla\boldsymbol{\varepsilon}_{\mathrm{SPE}}\right) \tag{4.5b}$$

$$\boldsymbol{h}_{\mathrm{SPE}} = f_{\boldsymbol{h}_{\mathrm{SPE}}}\left(c_{\mathrm{SPE}}, \nabla c_{\mathrm{SPE}}, \nabla\boldsymbol{\varepsilon}_{\mathrm{SPE}}\right) \tag{4.5c}$$

$$\boldsymbol{\sigma}_{\mathrm{SPE}} = f_{\boldsymbol{\sigma}_{\mathrm{SPE}}}\left(c_{\mathrm{SPE}}, \boldsymbol{\varepsilon}_{\mathrm{SPE}}\right) \tag{4.5d}$$

where $\boldsymbol{\varepsilon}_{\mathrm{SPE}}$ is the symmetric part of the gradient of $\boldsymbol{u}_{\mathrm{SPE}}$. Functions $f_{\boldsymbol{i}_{\mathrm{el}}}$, $f_{\boldsymbol{i}_{\mathrm{SPE}}}$, $f_{\boldsymbol{h}_{\mathrm{SPE}}}$, and $f_{\boldsymbol{\sigma}_{\mathrm{SPE}}}$ represent the constitutive equations used in the model, and are reported in Refs. [17, 18].

Finally, we ensure that the lithium content in the SPE at steady state charge/discharge conditions equals the lithium content at rest (*i.e.*, the lithium content in the SPE is preserved during charge/discharge processes). To this end, we enforce

$$\frac{1}{V_{\mathrm{SPE}}} \int_{V_{\mathrm{SPE}}} c_{\mathrm{SPE}} \,\mathrm{d}V = c^0, \tag{4.6}$$

where constant c^0 is the (uniform) concentration of lithium in the SPE when no current flows (salt concentration in the SPE, according to terminology used in Refs. [17,

Target field name	Field notation	Unit	Dimension
strain	$[\boldsymbol{\varepsilon}_{xx}, \boldsymbol{\varepsilon}_{yy}, \boldsymbol{\varepsilon}_{xy}]$	-	$n \times 3$
stress	$[\boldsymbol{\sigma}_{xx}, \boldsymbol{\sigma}_{yy}, \boldsymbol{\sigma}_{zz}, \boldsymbol{\sigma}_{xy}]$	MPa	$n \times 4$
electrostatic potential	$\boldsymbol{\phi}$	V	$n \times 1$
electric current density	$[\boldsymbol{i}_x, \boldsymbol{i}_y]$	$\mathrm{A\,m^{-2}}$	$n \times 2$
concentration	$\boldsymbol{c}$	$\mathrm{mol\,\mu m^{-3}}$	$n \times 1$
mass flux	$[\boldsymbol{h}_x, \boldsymbol{h}_y]$	$\mathrm{mol\,m^2\,s}$	$n \times 2$

Table 4.1: The target fields. The last column lists the dimensions of the arrays used to store the fields listed in the first column in a plane strain FE analysis. The number of nodes in the FE model is n.

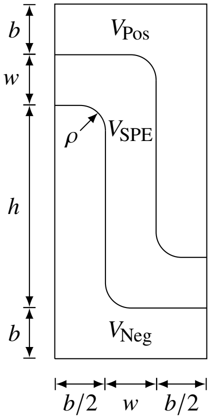


Figure 4.1: Schematic of a slice of the all-solid-state micro-battery with trench geometry.

18]). Constraint (4.6) guarantees the uniqueness of the solution of the system of equations (4.1-4.5). The description of boundary conditions and interactions between cell components (that makes the governing equations of electrodes and SPE coupled) is provided in Ref. [17]. We stress that no external applied displacement have been considered in this study (i.e., $\Delta u = 0$, according to the notation used in Ref. [17, Eq. (32)]).

The set of coupled equations (4.1-4.5) is solved using a standard FEM (refer to [17, Appendix A] for details about the time-dependent implementation). Considering a two dimensional model and plane strain conditions, the outcome of the FEM analysis is a set of fields defined at the nodes of the finite element mesh. The fields relevant for this study are listed in Table 4.1. These fields are either primary or secondary solution fields, i.e., solutions of the coupled set of PDEs or generated through post-processing, respectively. Since we are primarily interested in predicting these fields using an efficient surrogate, we refer to them as 'target fields'. The governing equations are solved using the material properties reported in Table 3 of Grazioli et al. [17] by setting E = 500 MPa, and Ω = 1.5×10^{-4} $\mathrm{m^3 mol^{-1}}$. In this study we only allow geometric parameters of the specimen in Fig. 4.1 to vary.

4.3. TACKLING THE BOTTLENECKS OF A FEM ANALYSIS WITH A DATA DRIVEN SURROGATE

We focus on the prediction of the target fields in Table 4.1 for a given battery cell geometry using an efficient surrogate model. The computational bottleneck of the aforementioned FEM analysis is due to two processes: 1) the inversion of the system matrix, and 2) the evaluation of nonlinear constitutive models at the integration points. To overcome these bottlenecks, we propose replacing the FEM analysis with a data-driven surrogate whose evaluation does not require a matrix inversion or computationally expensive nonlinear evaluations. Specifically, we propose a convolutional neural network (CNN)-based surrogate. The CNN-based surrogate takes in the geometry of the battery cell and generates an approximation of the target fields.

It is of interest to realize that the most computationally expensive operation in a CNN model is a set of convolution operations. A convolution operation can be performed as matrix multiplication [19], and its computational cost is therefore negligible when compared to matrix inversion or the evaluation of a nonlinear constitutive model. Concretely, the CNN model performs a pixel-to-pixel regression task in which a set of continuous input fields on a structured grid is transformed to another set of continuous fields on the same grid. Input fields should encode information about the battery cell geometry and output fields are an approximation of target fields. In the following sections, we first propose a set of methods to encode the geometry of a battery cell into a set of continuous fields and then discuss the interpolation of solution fields from FE nodes onto the same grid. Afterwards, we train the CNN-based surrogate using a supervised learning algorithm. In other words, we consider a set of battery cell specimens with different geometries and we compute their respective solution fields through FEM as summarized in the previous section. We then optimize the CNN-based surrogate parameters in order to minimize the difference between the output of the surrogate and that of the FEM analysis.

4.4. PREPROCESSING

As anticipated in the previous section, the CNN-based surrogate only accepts image-like fields on a structured grid (this will be discussed in Section 4.5). We therefore need to describe the geometry of the battery cell in such a format. The geometry of a battery cell consists of a set of attributes, i.e., domains and boundaries, defined at specific coordinates. In Section 4.4.2, we encode these attributes into image-like fields, such that each field describes a certain geometric attribute and its respective location.

The output of the CNN-based surrogate is also a set of image-like fields defined on a structured grid. Target fields however are outputs of the FEM analysis and are defined either on FE nodes or on Gauss points (and in the latter case the fields are, for convenience, extrapolated to nodes). This poses a difficulty when training the CNN-based surrogate. Training is essentially an iterative optimization procedure, where at each step the error between surrogate outputs and target fields is evaluated. To evaluate this error, both surrogate outputs and target fields have to be defined on either FE nodes or the structured grid. If FE nodes are chosen, the surrogate output should be interpolated at every training iteration. Since such an interpolation is a computationally expensive operation, as discussed in Section 4.4.1, the training phase will be severely affected in terms of numerical efficiency. To overcome this issue, target fields are interpolated onto the structured grid once, before the training starts, and reused during the training.

4.4.1. Interpolating target fields from an unstructured mesh onto a structured grid

We consider an equidistant rectangular grid of width v and height h, with n_x and n_y grid points in the horizontal and the vertical directions, respectively. Values of v and h should be chosen such that the grid spans over the entire geometry of the battery cell, as depicted in Fig. 4.1. The resolution of the grid, i.e., v/n_x or h/n_y (since the grid is equidistant, v/n_x and h/n_y are equal), should be fine enough to capture details of the geometry and target fields, but not unnecessarily too fine to avoid increasing the computational cost of the CNN-based surrogate. As such, the resolution is a problem-dependent parameter that could be defined by an automatic procedure. In this work a simple heuristic approach (i.e., visual inspection) has been employed, resulting in the use of a $n_y \times n_x = 512 \times 64$ grid which is fine enough to capture all details across all target fields.

After the definition of the grid, we interpolate target fields from FE nodes onto it. To do so, we follow the procedure by Silva et. al. [20] that is briefly illustrated for completeness using an example and with reference to a generic target field indicated by $\boldsymbol{\theta}$. Note that this field indicates a quantity defined at FE nodes. Quantities evaluated at Gauss points, such as the strain field, should be first extrapolated to FE nodes using standard FEM procedures. To interpolate field $\boldsymbol{\theta}$ from FE nodes onto a grid point g, we first identify the element e (in the FE mesh) in which the grid point g lies. For that we make use of the `find_simplex` function from the SciPy library [21]. We then extract values $\boldsymbol{\theta}^e$ at the element nodes and evaluate the element e's shape functions at the grid point g, $\boldsymbol{N}^e(\boldsymbol{x}^g)$, where $\boldsymbol{x}^g$ is the coordinate of the grid point g. The interpolated field at the grid point g is finally given by $\boldsymbol{\theta}^g = \boldsymbol{N}^e(\boldsymbol{x}^g)\ \boldsymbol{\theta}^e$. This procedure is computationally expensive and is only carried out during preprocessing and before training the CNN-based surrogate, not during its application in a multi-query application.

4.4.2. Encoding the geometry into a set of image-like fields

The geometry of a battery cell specimen, schematically depicted in Fig. 4.1, is encoded into a set of image-like fields. We refer to these fields as 'geometric fields'. Specifically, we discuss three types of geometric fields: (1) the 'distance-to-boundary' field, which contains information regarding the distance of a point to a certain boundary segment, (2) the 'is-on-boundary' field, which indicates whether a point is located on a boundary segment, and (3) the 'is-in-domain' field, which identifies the domain on which the point is located. These three simple geometric fields, depicted in Fig. 4.2 and explained next, have been selected to demonstrate the concept of geometric encoding. Their effectiveness is demonstrated in Section 4.7.4, and other encodings could be easily defined to improve the accuracy of the CNN surrogate.

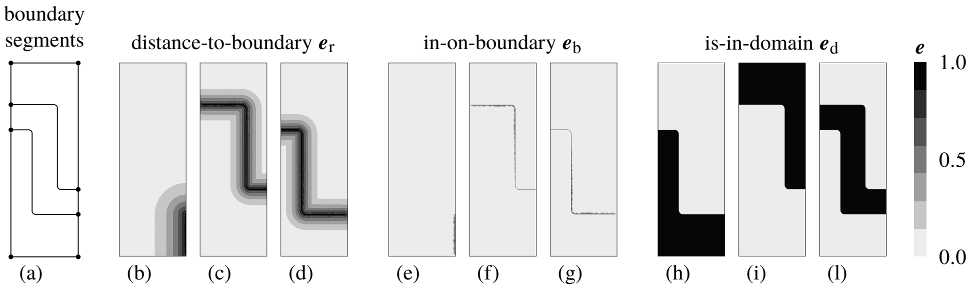

boundary segments
distance-to-boundary $\boldsymbol{e}_\mathrm{r}$
in-on-boundary $\boldsymbol{e}_\mathrm{b}$
is-in-domain $\boldsymbol{e}_\mathrm{d}$
$\boldsymbol{e}$
1.0
0.5
0.0
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
(l)

Figure 4.2: The three geometric fields used in this in this study for a 46 μm × 16 μm (height × width) specimen. Panels (b) to (d) show the distance-to-boundary fields for three of the ten boundary segments of the specimen in Fig. 4.1. A boundary segment is any of the lines connecting two consecutive dots in panel (a). The value of κ in Eq. (4.8) is equal to 0.1. These fields endow each point in the geometry with information regarding its closeness to the corresponding boundary segment. The closer the value of the field to one, the closer the point is to the boundary. Panels (e) to (f) show the is-on-boundary fields for three of the ten boundary segments of the specimen in Fig. 4.1. These fields endow each point in the geometry with information regarding its location with respect to corresponding boundary segment. If the value is equal to one, the point is located on that specific boundary segment. Panels (h) to (l) show the is-in-domain fields. These fields endow each point in the geometry with information regarding its location with respect to a domain (positive, or negative electrode, or SPE). If the value of the field is equal to one, the point is located on that specific domain.

DISTANCE-TO-BOUNDARY FIELD

The distance-to-boundary field measures the closeness of a point to a certain boundary segment. Let us assume that a boundary segment b is defined by $n^{[b]}$ points at coordinates $\boldsymbol{X}^{[b]}$. The closest Euclidean distance between a grid point i to the boundary b is given by

$$r_i^{[b]} = \min_j \|\boldsymbol{X}_i - \boldsymbol{X}_j^{[b]}\|_2 \quad \text{with} \quad j = 1, ..., n^{[b]}, \tag{4.7}$$

where j is a point on the boundary b and $\boldsymbol{X}_i$ contains the coordinates of grid point i. Equation (4.7) can be evaluated in a vectorized manner using the pairwise distance function `cdist` in the Scipy library [21]. The result of this computation is a continuos field $\boldsymbol{r}^{[b]} \in \mathbb{R}^{n_y \times n_x}$ with values in the semi-infinite domain $[0,\infty)$. For numerical convenience, values in $\boldsymbol{r}^{[b]}$ are transformed using the exponential function

$$\boldsymbol{e}_\mathrm{r}^{[b]} = \exp\left(-\frac{\boldsymbol{r}^{[b]}}{\kappa}\right), \tag{4.8}$$

where κ is a scaling factor, and $\boldsymbol{e}_\mathrm{r}^{[b]} \in [0,1]$ is the distance-to-boundary field. An example of this field is depicted in Fig. 4.2(b to d). The value of $\boldsymbol{e}_\mathrm{r}^{[b]}$ at a grid point being close to one implies that the grid point is close to the boundary b.

The purpose of distance-to-boundary fields is to encode a specific attribute of the geometry, namely the location of boundaries and points that are close to boundaries. If κ is very large, $\boldsymbol{e}_\mathrm{r}^{[b]}$ becomes close to one over the entire domain and hence, it cannot pinpoint the location of boundaries. If κ is very small, $\boldsymbol{e}_\mathrm{r}^{[b]}$ becomes close to zero over the entire domain except along the boundary, and hence it fails to identify points which are close to the boundary. An appropriate value of κ can be identified through visual inspection. For instance, $\kappa = 0.1$ satisfies the aforementioned requirement in Fig. 4.2(b to d).

IS-ON-BOUNDARY FIELD

The is-on-boundary field $\boldsymbol{e}_{\mathrm{b}}^{[b]} \in \mathbb{R}^{n_y \times n_x}$ identifies boundary segment b in a specimen. Closest grid points to a specific boundary segment get the value of one while the others are set to zero.

To compute this field, one could identify the boundary segment b by selecting $n^{[b]}$ number of points on it. Then, the closest grid point to the point j on boundary b is

$$p_j = \arg\min_i \|\boldsymbol{X}_i - \boldsymbol{X}_j^{[b]}\|_2 \quad \text{with} \quad i = 1, \ldots, n_y \times n_x, \tag{4.9}$$

where $n_y \times n_x$ is the total number of grid points. An example of this field is depicted in Fig. 4.2(e to g).

IS-IN-DOMAIN FIELD

The battery cell problem introduced in Section 4.2 consists of three domains; positive and negative electrode, and solid polymer electrolyte. The physics of each domain is governed by its own specific partial differential equations. To pass information about location of the domain d to the CNN-based surrogate, we construct the is-in-domain field $\boldsymbol{e}_{\mathrm{d}}^{[d]} \in \mathbb{R}^{n_y \times n_x}$. We follow the logic outlined in the previous section and identify the domain d in a field by setting values of grid points in that domain to one and the rest of the grid points to zero as shown in Fig. 4.2(g to i).

4.4.3. STACKING GEOMETRIC AND SOLUTION FIELDS WITH DIFFERENT SIZES

In the previous sections we explained how to preprocess a single battery cell specimen by encoding its geometry into image-like fields and interpolating the solution fields onto the same grid. However, we are primarily interested in preprocessing several specimens with different geometries and stack the resulting geometric and target fields into a single geometric tensor and a single solution tensor, respectively. We stack target fields of several specimens into the solution tensor $\boldsymbol{S} \in \mathbb{R}^{m \times n_y \times n_x \times n_{\mathrm{s}}}$, where m is the number of specimens, n_y and n_x are number of grid points in vertical and horizontal directions, and n_{s} is the number of target fields. Likewise, we stack geometric fields into the geometric tensor $\boldsymbol{E} \in \mathbb{R}^{m \times n_y \times n_x \times n_{\mathrm{g}}}$, where n_{g} is the number of geometric fields. Depending on which geometric fields one considers, the value of n_{g} differs. For instance, should one only consider distance-to-boundary and is-in-domain fields, then n_{g} becomes $n_{\mathrm{b}} + n_{\mathrm{d}}$, where n_b is the number of boundaries, and n_d is the number of domains.

Solution and geometric tensors collect values sampled over grids of the same $n_y \times n_x$ size, irrespective of the specimen geometry. This implies that when stacking several specimens with different geometries each specimen only covers part of the grid. Ideally, we would like to train the CNN-based surrogate only on parts of the grid covered by the specimen. To this end, we construct a set of masks that identify regions of the grid covered by each specimen. These masks are used in Section 4.7 to train the CNN-based surrogate only on the region actually covered by each specimen.

4.4.4. STANDARD SCALING

Up to this point, we discussed the construction of geometric, mask, and solution tensors. The first two are input to the CNN-based surrogate while the last is the target that we train the surrogate for. The training of a CNN-based surrogate, i.e., the optimization of its parameters, is sensitive to the magnitude of its inputs and targets [22]. We further elaborate on this aspect in Section 4.5.3. To boost the stability and the speed of the optimization process, according to LeCun [22], these values should have the mean and the standard deviation of approximately zero and one, respectively.

Let us assume that we have an m number of battery cell specimens. For each specimen we compute the geometric and the solution tensors. This process is referred to as data collection (further discussed in Section 4.7.1). Consider the geometric tensor $\boldsymbol{E}$ (we apply the same process for the solution tensor $\boldsymbol{S}$). To avoid overfitting and to increase generalization of the CNN-based surrogate, we train it only on some specimens and then test its generalization capabilities on the others. Concretely, we divide specimens into training $\boldsymbol{E}^{\text{train}} \in \mathbb{R}^{m_{\text{train}} \times n_y \times n_x \times n_{\text{g}}}$ and test $\boldsymbol{E}^{\text{test}} \in \mathbb{R}^{m_{\text{test}} \times n_y \times n_x \times n_{\text{g}}}$ tensors, where $m = m_{\text{train}} + m_{\text{test}}$. Next, we compute the mean $\boldsymbol{\mu}^{\text{train}} = \mathbb{E}(\boldsymbol{E}^{\text{train}}) \in \mathbb{R}^{n_{\text{g}}}$ and the standard deviation $\boldsymbol{s}^{\text{train}} = \sqrt{\mathbb{E}(\boldsymbol{E}^{\text{train}} - \boldsymbol{\mu}^{\text{train}})^2} \in \mathbb{R}^{n_{\text{g}}}$ of the training tensor, where $\mathbb{E}$ indicates the expected value. With these quantities, we scale both training and test tensors according to

$$\begin{aligned} \boldsymbol{E}^{\text{train}} &\leftarrow \frac{\boldsymbol{E}^{\text{train}} - \boldsymbol{\mu}^{\text{train}}}{\boldsymbol{s}^{\text{train}}}, \\ \boldsymbol{E}^{\text{test}} &\leftarrow \frac{\boldsymbol{E}^{\text{test}} - \boldsymbol{\mu}^{\text{train}}}{\boldsymbol{s}^{\text{train}}}. \end{aligned} \tag{4.10}$$

This procedure is referred to as standard scaling. To avoid data leakage [23], it is essential to scale the test tensor using the mean and standard deviation of the training tensor. Data leakage occurs when the performance of the model on the test data is considerably better than that of the real-life due to the inclusion of illegitimate data [23]. Illegitimate data is what we do not have access to in real life. In this case, we do not have access to the mean and standard deviation of the test set in a real-life scenario.

It is of importance not to scale the mask tensor $\boldsymbol{M}$ because it loses its binary format and therefore fails to discriminate between parts of the grid that the specimen does not cover and parts that it does.

4.4.5. SUMMARY OF PREPROCESSED FIELDS

Before moving on to the definition of the CNN-based surrogate, let us summarize the preprocessing stage. We consider m differently-shaped battery cell specimens. For each specimen, we compute target fields through a FEM analysis as discussed in Section 4.2. Afterwards, we construct a structured grid that is large enough to accommodate all specimens. To describe the geometry of each battery cell as image-like fields, we construct the geometric tensor on the aforementioned grid. We also interpolate target fields onto the same grid and stack them into a solution tensor. Finally, we divide specimens into a training and a test tensors and scale them according to the procedure in Section 4.4.4.

Table 4.2 lists geometric, target, and mask fields, together with their respective units and dimensions. Tensors $\boldsymbol{E}$ and $\boldsymbol{M}$ are inputs the to CNN-based surrogate. The opti-

	Field name	Field notation	Unit	Dimension
Geometric ($\boldsymbol{E}$)	distance-to-boundary	$\boldsymbol{e}_\mathrm{r}$	-	$m \times n_y \times n_x \times n_\mathrm{b}$
	is-on-boundary	$\boldsymbol{e}_\mathrm{b}$	-	$m \times n_y \times n_x \times n_\mathrm{b}$
	is-in-domain	$\boldsymbol{e}_\mathrm{d}$	-	$m \times n_y \times n_x \times n_\mathrm{d}$
Target ($\boldsymbol{S}$)	strain	$[\boldsymbol{\varepsilon}_{xx}, \boldsymbol{\varepsilon}_{yy}, \boldsymbol{\varepsilon}_{xy}]$	-	$m \times n_y \times n_x \times 3$
	stress	$[\boldsymbol{\sigma}_{xx}, \boldsymbol{\sigma}_{yy}, \boldsymbol{\sigma}_{zz}, \boldsymbol{\sigma}_{xy}]$	MPa	$m \times n_y \times n_x \times 4$
	electrostatic potential	$\boldsymbol{\phi}$	V	$m \times n_y \times n_x \times 1$
	electric current density	$[\boldsymbol{i}_x, \boldsymbol{i}_y]$	$\mathrm{A\,m^{-2}}$	$m \times n_y \times n_x \times 2$
	concentration	$\boldsymbol{c}$	$\mathrm{mol\,\mu m^{-3}}$	$m \times n_y \times n_x \times 1$
	mass flux	$[\boldsymbol{h}_x, \boldsymbol{h}_y]$	$\mathrm{mol\,m^2\,s}$	$m \times n_y \times n_x \times 2$
Mask ($\boldsymbol{M}$)		-	-	$m \times n_y \times n_x \times 1$

Table 4.2: Preprocessed field on a $n_y \times n_x$ grid. The numbers of boundaries and domains are n_b and n_d, respectively, m is the number of specimens, n_x and n_y are the numbers of grid points in the horizontal and vertical directions, respectively, and solution fields are those of Table 4.2 but interpolated onto the $n_y \times n_x$ grid.

mization of the parameters of the CNN-based surrogate such that its outputs becomes acceptably close to the target (i.e., tensor $\boldsymbol{S}$) is discussed in Section 4.5.3.

4.5. CONVOLUTIONAL NEURAL NETWORK-BASED SURROGATE

As discussed in Section 4.3, the typical FEM analysis conducted in this study has two computational bottlenecks: evaluation of a nonlinear constitutive model and inversion of the system matrix. In this section we introduce HydraNet, a convolutional neural network variant that is computationally independent of physics-based constitutive models and whose evaluation does not require a matrix inversion.

4.5.1. BUILDING BLOCKS OF A CONVOLUTIONAL NEURAL NETWORK

A convolutional neural network is essentially a stack of several different functions. These functions are the building blocks of the convolutional neural network and are commonly referred to as layers. Each layer takes the output of the previous layer, transforms it in a certain manner, and passes the output to the next layer.

CONVOLUTION LAYER

The convolution layer is comprehensively explained in Ref. [19]. Here we introduce its essential parts. The convolution layer, unlike what its name suggests, is a cross-correlation operation. This operation cross-correlates the kernel tensor $\boldsymbol{W}_\mathrm{c} \in \mathbb{R}^{k \times k \times f^\mathrm{i} \times f^\mathrm{o}}$ over the input of the convolution layer $\boldsymbol{I} \in \mathbb{R}^{m \times n_y^\mathrm{i} \times n_x^\mathrm{i} \times f^\mathrm{i}}$ to generate the output of the convolution layer $\boldsymbol{O} \in \mathbb{R}^{m \times n_y^\mathrm{o} \times n_x^\mathrm{o} \times f^\mathrm{o}}$:

$$\boldsymbol{O} = \boldsymbol{W}_\mathrm{c} * \boldsymbol{I} + \boldsymbol{b}_\mathrm{c}. \tag{4.11}$$

In the equation above, $*$ is the cross-correlation operation, f^i and f^o are the number of input and output filters, respectively, the input and output tensors are defined on $n_y^\mathrm{i} \times n_x^\mathrm{i}$

and $n_y^{\mathrm{o}} \times n_x^{\mathrm{o}}$-sized grids, respectively, k is the size of the kernel tensor, and $\boldsymbol{b}_{\mathrm{c}} \in \mathbb{R}^{f^{\mathrm{o}}}$ is the bias vector.

The output of the cross-correlation operation (4.11) has a smaller dimension than its input. For instance, if the input height n_y^{i} is 100 and the kernel size k is 5, the output height n_y^{o} becomes 96 (for details of this calculation refer to the spatial arrangement section in Ref. [19]). To avoid this undesirable dimension reduction, it is common to pad perimeters of the input tensor with zeros in order to increase its dimension to $(n_y^{\mathrm{i}}+2p) \times (n_x^{\mathrm{i}}+2p)$, where p is the number of zero padded grid points on each side. In the previous example, the dimension of the input (i.e., $n_y^{\mathrm{o}} = n_y^{\mathrm{i}} = 100$) is preserved with $p = 1$.

If several convolution layers are stacked while preserving tensor dimensions, the computational complexity of the CNN becomes very large. As a remedy to this problem, it is common to reduce the dimension of the input tensor in a controlled manner through downsampling. In the cross-correlation operation, the kernel weight $\boldsymbol{W}_{\mathrm{c}}$ strides through the input tensor one grid point at a time. One could downsample the input tensor by simply striding the kernel weight by $s > 1$ points at a time (with $s = 2$, the output dimension n_y^{o} becomes 50, i.e., half of the dimension of the input field). This process is clearly illustrated in Ref. [24]. For detail of this computation, refer to the spatial arrangement section of Ref. [19]. A consequence of the downsample by striding just discussed is an unwanted size unbalance. Should we pass the geometric tensor through a stack of convolution layers with $s > 1$, the dimension of the last output tensor becomes considerably smaller than that of the geometric tensor. However, as discussed in Section 4.3, the output of the CNN-based surrogate should be equal to the dimension of the grid that the solution field is defined on. Therefore, after a series of downsampling operations using the convolution layer with $s > 1$, we need to upsample the field to the initial grid size as discussed next.

Transposed convolution layer

The transposed convolution layer, as the name suggests, is the transposed version of the cross-correlation operation in the convolution layer. This operation is descriebed in Ref. [25, section 4.2] and it is clearly illustrated in Ref. [24, Fig. 13.10.1]. Essentially, a transposed convolution layer, with $s > 1$, upsamples the input field. Just like the convolution layer, the transposed convolution layers has a kernel weight $\boldsymbol{W}_{\mathrm{tc}}$ and a bias vector $\boldsymbol{b}_{\mathrm{tc}}$.

Let us follow up with the example in the previous section where we passed an input tensor with a height of 100 to a convolution layer with $s = 2$ and $p = 1$. The convolution layer reduces the height of the tensor to 50. To upsample the output tensor back to the height of 100, we could pass it to a transposed convolution layer with $s = 2$ and $p = 1$.

Both convolution and its transpose are linear operations. Stacking a number of these linear layers results in a linear CNN model. To add nonlinearity, we transform outputs of these layers with the rectified linear unit discussed in the next section.

Rectified linear unit

The convolution and the transposed convolution layers are linear operations. Stacking a number of these linear layers trivially results in another linear layer. To add nonlinearity to the CNN model, the output of each (transposed) convolution layer should be nonlinearly transformed. One nonlinear transformation that has been empirically proven to be effective is the rectified linear unit (ReLU) [26]:

$$\boldsymbol{O} = \max(\mathbf{0}, \boldsymbol{I}), \tag{4.12}$$

where $\boldsymbol{I}$ and $\boldsymbol{O}$ are the input and output tensors, respectively. This transformation returns zero where the input tensor has a negative value and the value of the input field otherwise.

There is a possibility that the distribution of the input tensor gets shifted to negative values. This phenomenon is referred to as internal covariate shift. In that scenario, the rectified linear unit trivially returns a zero tensor. As a remedy to this issue, batch normalization [27] is used to standardize input values such that their mean becomes zero and the standard deviation becomes one. In this contribution, however, we choose not to make use of the batch normalization layer as we observed that it does not significantly improve the accuracy of our model and it substantially increases the computational expense.

4.5.2. HYDRANET: A CONVOLUTIONAL NEURAL NETWORK FOR MULTI-PHYSICS PROBLEMS

The convolution layer, its transpose, and the rectified linear unit are stacked together to construct a CNN-based surrogate that we call HydraNet. We propose HydraNet as a surrogate for multi-physics FEM analysis. The architecture of HydraNet is depicted in Fig. 4.3. HydraNet consists of two parts: the body and a number of heads. The body transforms the geometric tensor $\boldsymbol{E}$ and passes the resulting tensor to the heads. Each head transforms the output of the body to a set of solution fields.

Let us analyze HydraNet by walking through its computation graph. Initially, the geometric tensor $\boldsymbol{E}$ goes through n_l convolution layers, each followed by a rectified linear unit. The value of the strides s in these convolution layers is equal to two. These operations are shown with black circles in Fig. 4.3. We refer to this part of HydraNet as the body. The rest of HydraNet consists of several heads. Each head transforms the output of the body to a set of solution fields. In each head, the output of the body is passed to n_l transposed convolution layers, with stride $s = 2$, each of which is followed by a rectified linear unit. These operations are indicated with gray circles in Fig. 4.3. The output of the aforementioned operations is then passed to a convolution layer, with stride $s = 1$, indicated by the white circle with a solid black line in Fig. 4.3. The number of filters in this convolution layer is equal to the cardinality of the subset of solutions associated with the head. For instance, if the subset of solutions is $[\hat{\boldsymbol{\varepsilon}}_{xx}, \hat{\boldsymbol{\varepsilon}}_{yy}, \hat{\boldsymbol{\varepsilon}}_{xy}]$, then the number of filters in the last convolution layer is equal to three. In the final step, all predicted solution fields are concatenated into the predicted solution tensor $\hat{\boldsymbol{S}}$. We indicate the concatenation operation by the dashed circle in Fig. 4.3. The $\hat{\boldsymbol{S}}$ tensor consists of predicted solution fields that span over the entire grid. However, each specimen only covers a part of the grid. We set parts of solution fields that are outside the specimen to zero by an element-wise multiplication of $\hat{\boldsymbol{S}}$ by the mask tensor $\boldsymbol{M}$. This operation is indicated by the dotted dashed circle in Fig. 4.3.

Hydranet's performance depends on the number of heads per solution field: on the one extreme, one could consider only one head for all the solution fields, on the other extreme, one head per solution field could be used. In the former scenario, the number

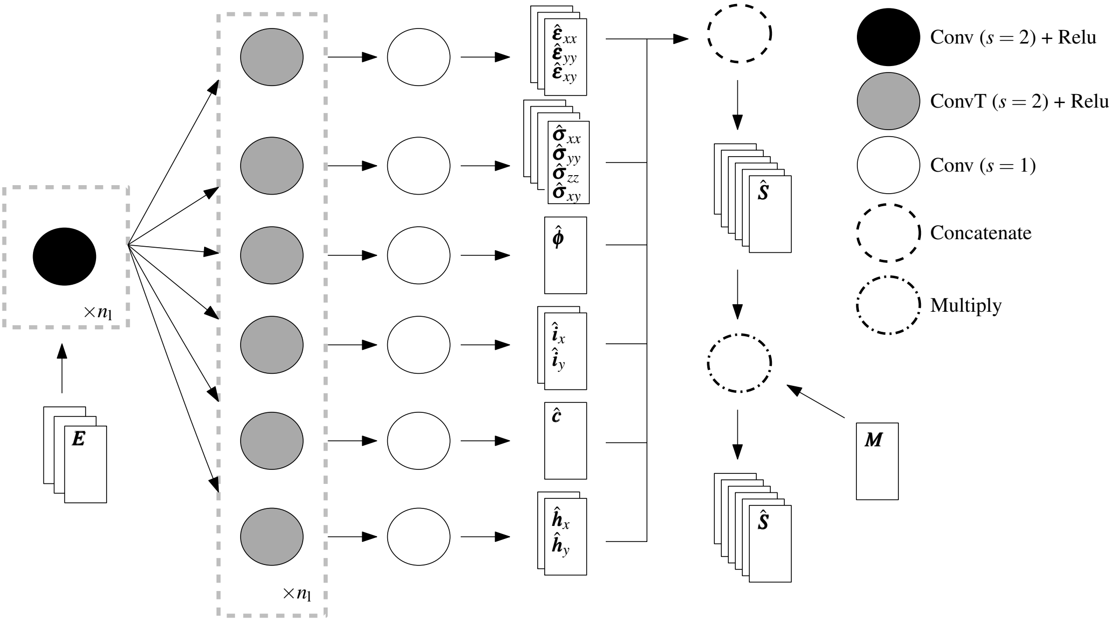


Figure 4.3: HydraNet consists of two parts: the body, which transforms the geometric tensor $\boldsymbol{E}$, and a number of heads, which transform the output of the body to the solution tensor. Conv ($s = 2$) + Relu stands for convolution layer (with stride $s = 2$) followed by the rectified linear unit. ConvT refers to the transposed convolution layer.

of kernel tensors and bias vectors is relatively small and, therefore, the capacity of HydraNet is limited, albeit the model runs the fastest. In the latter scenario, the number of kernel tensors and bias vectors is considerably large, resulting in a slow HydraNet characterized by a high capacity. Here, we choose a middle ground and consider one head per strain $[\hat{\boldsymbol{\varepsilon}}_{xx}, \hat{\boldsymbol{\varepsilon}}_{yy}, \hat{\boldsymbol{\varepsilon}}_{xy}]$, stress $[\hat{\boldsymbol{\sigma}}_{xx}, \hat{\boldsymbol{\sigma}}_{yy}, \hat{\boldsymbol{\sigma}}_{zz}, \hat{\boldsymbol{\sigma}}_{xy}]$, electrostatic potential $\hat{\boldsymbol{\phi}}$, electric current density $[\hat{\boldsymbol{i}}_x, \hat{\boldsymbol{i}}_y]$, concentration $\hat{\boldsymbol{c}}$, and mass flux $[\hat{\boldsymbol{h}}_x, \hat{\boldsymbol{h}}_y]$ fields.

Kernel weights and bias vectors are initially filled with random numbers and zeros, respectively. As a result, the predicted solution tensor $\hat{\boldsymbol{S}}$ is completely different than the target solution tensor $\boldsymbol{S}$. In the next section, we discuss how to tune kernel tensors and bias vectors to decrease the discrepancy between $\hat{\boldsymbol{S}}$ and $\boldsymbol{S}$.

4.5.3. OPTIMIZING THE PARAMETERS OF A CONVOLUTIONAL NEURAL NETWORK

HydraNet is essentially a function with a set of trainable parameters, i.e., kernel tensors and bias vectors. We expect this function to transform a geometric tensor $\boldsymbol{E}$ to a solution tensor $\hat{\boldsymbol{S}}$, such that the discrepancy between $\hat{\boldsymbol{S}}$ and the target solution tensor $\boldsymbol{S}$ is small.

As a measure of discrepancy between $\hat{\boldsymbol{S}}$ and $\boldsymbol{S}$, we define the mean squared error cost function

$$e = \frac{1}{m_{\text{train}}} \|\hat{\boldsymbol{S}} - \boldsymbol{S}\|_{\text{f}}^2, \tag{4.13}$$

where $\|\#\|_{\text{f}}$ is the Frobenius norm of # and m_{train} is the number of specimen in the training data.

We tune weight tensors and bias vectors to minimize the cost function (4.13) using a gradient-based optimizer called Adam [28, 29]. Among the parameters of the Adam optimizer, the learning rate is arguably the most important one. The learning rate determines the size of the steps that the gradient-based optimizer takes while moving towards a local minimum. We discuss the impact of the learning on the accuracy of HydraNet in Section 4.7.3. For the rest of the parameters, we make use of those suggested by Kingma and Ba [28].

The optimization procedure, regardless of the specific optimizer, has two other parameters: the number of epochs and the batch size. An epoch is one training cycle through the entire dataset; the batch size is the number of specimens passed to the optimizer at each training step. For instance, with 100 specimens and a batch size equal to 20, five batches should be sent to the optimizer until one epoch is over.

4.6. Software and hardware

HydraNet is implemented and trained on a GPU using the TensorFlow library [30]. TensorFlow includes implementation of several building blocks of a convolutional neural network and training functionalities such as the Adam optimizer.

We carry out the preprocessing stage, discussed in Section 4.4, in the Python programming language. To facilitate linear algebra operations, we make use of the SciPy library [21]. The interpolation procedure in Section 4.4.1 is the most computationally demanding part of the preprocessing stage and is parallelized using the Joblib library [31].

HydraNet can be downloaded from `https://github.com/FGhavamian/cnn_battery`.

4.7. Results and discussion

In this section we apply HydraNet to the battery cell problem presented in Section 4.2. We train several HydraNets with different optimization parameters, preprocessing methods, and architectures in Sections 4.7.3, 4.7.4, and 4.7.5, respectively, and we compare their respective accuracy using an evaluation metric in Section 4.7.2. Finally, in Section 4.7.6 we study the impact of the amount of training data on HydraNet's accuracy.

The parameters of a machine learning process that are set before the optimization process are referred to as hyperparameters. The learning rate of the optimizer, the method of encoding geometries, the number of layers in HydraNet, and the amount of training data are hyperparameters. In the following sections, we specifically study the impact of these hyperparameters on the accuracy of HydraNet on the data that HydraNet was not trained on.

4.7.1. Data collection and preprocessing

The schematic of the battery cell specimen is depicted in Fig. 4.1. We fix two parameters of the geometry, $\rho = 1$ μm and $b = 10$ μm, and let h and w vary. Specifically, we consider $h \sim \mathscr{U}(20\ \mu\text{m}, 100\ \mu\text{m})$ and $w \sim \mathscr{U}(5\ \mu\text{m}, 20\ \mu\text{m})$ to be uniformly distributed and randomly sample 50 tuples (h_i, w_i), $i = \{1, \ldots, 50\}$. For each tuple, we construct one battery cell specimen and compute its solution fields using the FEM analysis described in Section 4.2.

We then define a grid that is large enough to fit each battery cell specimens. As discussed in Section 4.4.1, we consider a 512 × 64 grid which has a height of 137 μm and a width of 30 μm. The geometry of each specimen is encoded into geometric fields according to Section 4.4.2. To keep track of the area of the grid covered by each specimen we make use of a mask tensor, as discussed in Section 4.4.3. Stacking all fields yields a geometric tensor $\boldsymbol{E} \in \mathbb{R}^{50\times512\times64\times n_g}$, a solution tensor $\boldsymbol{S} \in \mathbb{R}^{50\times512\times64\times13}$, and a mask tensor $\boldsymbol{M} \in \mathbb{R}^{50\times512\times64\times1}$. As mentioned in Section 4.4.5, n_g is the number of geometric fields, and there are in total 13 solution fields.

To check the generalization capabilities of HydaraNet, these tensors are randomly divided into training tensors $\boldsymbol{E}^{\text{train}}$, $\boldsymbol{S}^{\text{train}}$, $\boldsymbol{M}^{\text{train}}$ and test tensors $\boldsymbol{E}^{\text{test}}$, $\boldsymbol{S}^{\text{test}}$, $\boldsymbol{M}^{\text{test}}$. The training tensors are used to optimize the parameters of HydraNet while the test tensors are used to quantify the generalization capabilities of the trained model. Generalization refers to the ability of the model to make accurate predictions on the data it has not been trained on. As explained in Section 4.4.4, to speed up the training of HydraNet, we scale geometric tensors and solution tensors such that their mean and standard deviation become close to zero and one, respectively.

4.7.2. Evaluation metric

To compare two models with different hyperparameters, one must select a measure of performance. Such a measure is often referred to as the evaluation metric. The evaluation metric is a scalar that describes the performance of the model. Ideally, one should employ an evaluation metric that describes both accuracy and computational expense of the model. However, to establish such a metric one should consider the application's requirements and availability of the computing power. Therefore, for simplicity, we simply use a metric that describes the accuracy.

Since we are dealing with a regression problem, we employ the mean relative squared error

$$\text{MRSE} = \frac{\frac{1}{m}\|\boldsymbol{S}-\hat{\boldsymbol{S}}\|_F^2}{1+\frac{1}{m}\|\boldsymbol{S}\|_F^2} \tag{4.14}$$

as the evaluation metric. We add one to the denominator to avoid division by zero. The denominator is essentially a scalar that does not change when comparing two models with different hyperparameters. We also use scaled solution tensors, as discussed in Section 4.4.4, to evaluate (4.14).

Equipped with an evaluation metric and two sets of training and test tensors, in the following sections we discuss the impact of different hyperparameters on the accuracy of HydraNet.

4.7.3. Selection of the learning rate value

The learning rate is arguably by far the most important hyperparameter of a gradient-based optimizer. On the one hand, if it is very large the optimizer fails to converge to a local minimum; on the other hand, if the learning rate is very small the gradient-based optimizer takes an excessive amount of time to converge. In this section we analyze the impact of different learning rates on the accuracy of HydraNet. We train three HydraNets using learning rates $\eta \in \{0.01, 0.001, 0.0001\}$. We fix the remaining hyperparameters to

η	0.01	**0.001**	0.0001
MRSE × 10^2	6.5	**4.6**	9.7

Table 4.3: The mean relative squared error on test tensors at different learning rates η. It appears that $\eta = 0.001$ is the most efficient among the three.

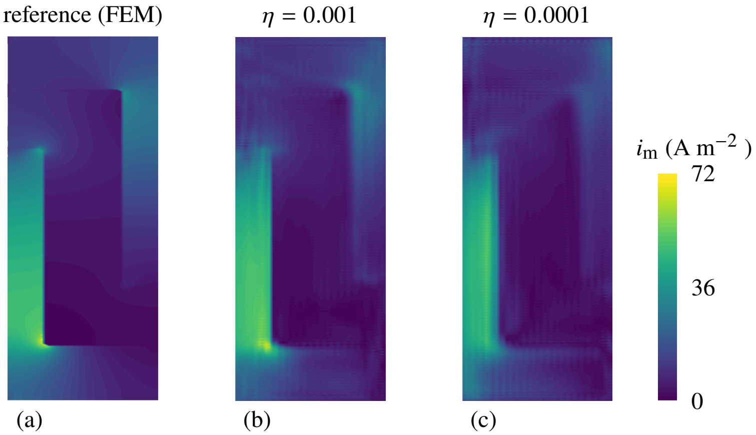


Figure 4.4: Magnitude of the electric current density $\boldsymbol{i}_\mathrm{m} = \sqrt{\boldsymbol{i}_x^2 + \boldsymbol{i}_y^2}$ predicted by (a) the FE model, (b) HydraNet with learning rate $\eta = 0.001$, and (c) HydraNet with learning rate $\eta = 0.0001$.

the following: number of layers $n_\mathrm{l} = 1$, kernel weight size $k = 7$, filter size $f_\mathrm{o} = 16$ for the convolution layer and its transpose, distance-to-boundary method to encode geometries of specimens, number of epochs equal to 200, and batch size equal to 16.

Table 4.3 reports the MRSE on test tensors. It is evident that the learning rate $\eta = 0.001$ is the most efficient among the three. On the one hand, it is large enough to let the optimizer descend rapidly to a local minimum, and on the other hand, it is small enough to ensure the stability of the optimizer.

For the sake of illustration, we randomly select a specimen from the test data and plot the magnitude $\boldsymbol{i}_\mathrm{m} = \sqrt{\boldsymbol{i}_x^2 + \boldsymbol{i}_y^2}$ of the electric current density field for learning rates η equal to 0.001 and 0.0001 in Fig. 4.4. The peak in the electric current density on the bottom left corner of the electrolyte is completely disregarded when $\eta = 0.0001$ and becomes evident by increasing η to 0.001. The issue with $\eta = 0.0001$ is that the optimizer descends to a local minimum at a much slower pace than when the learning rate is equal to 0.001. In other words, HydraNet is not trained enough. Allowing the optimization process to continue long after epoch 200, the accuracy of HydraNet with the learning rate $\eta = 0.0001$ will eventually catch up with that of Hydranet with $\eta = 0.001$. In the rest of this study we use 0.001 as an efficient learning rate value.

Geometry encoding method	MRSE × 10^2
is-on-boundary	18.3
is-on-domain	18.9
is-on-boundary + is-on-domain	14.6
distance-to-boundary	4.6
distance-to-boundary + is-on-domain	3.8
distance-to-boundary + is-on-boundary	3.8
distance-to-boundary + is-on-boundary + is-on-domain	**3.6**

Table 4.4: The mean relative squared error on the test set. It appears that the richer the features, the more accurate HydraNet becomes.

4.7.4. RICHNESS OF THE GEOMETRY ENCODING

The effectiveness of the methods described in Section 4.4.2 to encode the geometry of a specimen into image-like fields is evaluated. We train a HydraNet for each case using the values of the hyperparameters defined earlier. Table 4.4 reports the MRSE on the test tensors. The results indicate that the most effective method to encode a geometry is the distance-to-boundary method for which the MRSE is equal to 4.5×10^{-2}. In contrast, considering both is-on-domain and is-on-boundary fields yields a MRSE equal to 14.6×10^{-2}. It is also evident that HydraNet becomes the most accurate, with a MRSE equal to 3.6×10^{-2}, when geometric fields of all three methods are combined. This combination results in the most expensive HydraNet and is the one that we use in the rest of the study. Requirements of a specific application determine whether this combination is worth the computational effort. A more in-depth study of the contribution of each input falls into the realm of the interpretability of machine learning models (refer, e.g., to the work by Molnar [32]).

For the sake of illustration, Fig. 4.5 shows the von Mises stress distribution of a randomly selected specimen from the test data. Note that, according to Section 4.2, the von Mises stress is only defined on the electrolyte. This field is therefore equal to zero on other parts of the battery cell. Through visual inspection, it appears that the improvement as a result of considering all geometric fields over the distance-to-boundary fields is marginal. Nevertheless, we do observe local improvements. For instance, the boundary between the electrolyte and the positive electrode is blurry when only the distance-to-boundary is considered. This boundary becomes sharper when geometric fields describing domains and location of boundaries are also considered.

4.7.5. HOW DEEP OR WIDE SHOULD THE HYRANET BE?

The HydraNet model is essentially a nonlinear function. This nonlinear function models a relationship between the input and the output data. The more layers and the more filters in each layer, the more complex relationship HydraNet can model. Here, we study the impact of the numbers of layers and filters on the accuracy of HydraNet.

We consider the same number of layers and filters for the body and each head of HydraNet. We indicate the number of layers and filters of HydraNet using an array. For instance, $[16, 32, 64]$ refers to a HydraNet with three layers in the body and each head.

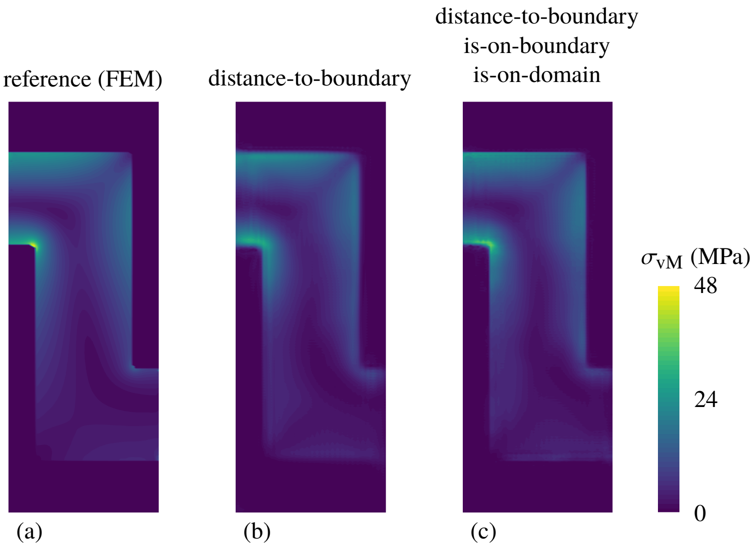


Figure 4.5: von Mises stress field obtained from the FE model (a), that of HydraNet when considering only distance-to-boundary fields (b), and that of HydraNet when considering all three types of geometric fields (c). It appears that the boundary between the electrolyte and the positive electrode is blurry when only the distance-to-boundary is considered. This boundary is sharper when geometric fields describing domains and location of boundaries are also considered.

Number of layers	1			2			3		
Number of filters	[8]	[16]	[32]	[8, 16]	[16, 32]	**[32, 64]**	[8, 16, 32]	[16, 32, 64]	[32, 64, 128]
MRSE × 10^2	5.1	3.6	3.0	3.9	3.0	**2.6**	3.5	2.9	2.9

Table 4.5: The mean relative squared error on the test set. It appears that a two-layer HydraNet is the most accurate. By increasing the number of filters HydraNet becomes more and more accurate.

The body has 16, 32, and 64 filters in the first, second, and third layer, respectively. And each head has 64, 32, and 16 filters in the first, second, and third layer, respectively. We set the learning rate to the previously tuned value of 0.001, encode the geometry using the combination of distance-to-boundary, is-on-domain, and is-on-boundary methods, and the rest of the hyperparameters are those reported in the previous sections. We train several HydraNets with a different number of layers and filters, compute their respective MRSE on the test set, and report the results in Table 4.5.

In a general setting, the impact of the number of layers and filters of HydraNet on its accuracy is governed by the bias-variance trade-off concept. At the one extreme, should the number of layers or filters be few, HydraNet fails to model complex relationships between the input and the output data (i.e., HydraNet becomes biased). At the other extreme, should the number of layers or filters be too many, HydraNet overfits the training data and is unable to generalize to unseen data (i.e., HydraNet has high variance). In both scenarios, HydraNet's accuracy diminishes.

For the problem at hand, the results reported in Table 4.5 clearly indicate that by increasing the number of filters HydraNet becomes more and more accurate. For instance, by increasing the number of filters from 8 to 32, the MRSE decreases by approximately 40%. This implies that the limit at which adding more filters would decrease HydraNet's accuracy is not yet reached. Nevertheless, adding more filters increases the computational cost. The balance between accuracy and computational cost is determined by the practical requirements of a use case.

It appears that adding more layers does not monotonically decrease the MRSE. For instance, the MRSE of a two-layer HydraNet with $[32, 64]$ filters has a MRSE of 2.6×10^{-2} on the test data. By adding another layer to this HydraNet, $[32, 64, 128]$, the MRSE increases by approximately 10%. There could be two explanations for this observation. First, the deeper HydraNet gets, the harder it becomes to train [33]. Second, HydraNet could have started to overfit the training data. The second explanation is more likely. The MRSE on the training data is 2.5×10^{-2} while that of on the test data is 2.9×10^{-2}. This implies that HydraNet is still capable to fit itself to the training data, but it is slowly losing its generalization capabilities.

For the sake of illustration, we randomly select a specimen from the test data and depict its concentration in Fig. 4.6. The MRSE of a two-layer HydraNet ($[32, 64]$) is lower than that a single-layer HydraNet ($[16]$) (the difference is 1.0×10^{-2}). This decrease in the MRSE value is also evident in Fig. 4.6. Compared to the figure in the middle, where there is only one layer with 16 filters, in the figure on the right, where the number of layers is increased to two and there are 32 and 64 filters, we observe that the concentration distribution is sharper and bears more resemblance to that of the FE reference. We therefore proceed with the most accurate HydraNet, i.e., the two-layer HydraNet with $[32, 64]$ filters.

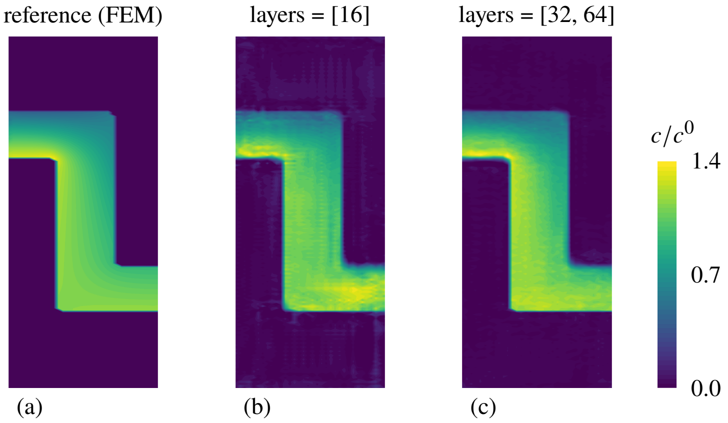


Figure 4.6: Normalized lithium concentration obtained from the FE model (a), that of HydraNet when considering only distance-to-boundary fields (b), and that of HydraNet when considering all three types of geometric fields (c). Compared to panel (b), where there is only one layer with 16 filters, the results in panel (c), where the number of layers is increased to two and there are 32 and 64 filters, we observe that the concentration distribution is sharper and bears more resemblance to that of the FE reference.

Percentage of all training data	25	50	**75**	100
MRSE on test data $\times 10^2$	4.5	2.9	**2.6**	2.6

Table 4.6: The mean relative squared error on the test set. It appears that the generalization capability of the model increases with the amount of data that is used during training up to a 75% of current training data. Afterward, adding more training data has little impact on improving the MRSE on the test data.

4.7.6. IMPACT OF THE QUANTITY OF TRAINING DATA ON ACCURACY

Next, we determine whether adding more training data helps increasing the generalization capability of HydraNet. We set the learning rate to 0.001, encode the geometry using the combination of distance-to-boundary, is-on-domain, and is-on-boundary methods, consider a two-layer HydraNet ([32,64]), and set the rest of the hyperparameters to the values used in the previous sections.

Collecting more data requires executing a computationally expensive FEM analysis several times. It is often a good idea to a priori test the impact of adding more training data on the generalization capabilities by means of an examination of the learning curve [34]. The learning curve shows how the MRSE on the test data changes by increasing the training data. For the problem at hand, we conducted such a study by considering four scenarios: training HydraNet with 25%, 50%, 75%, and 100% of the training data. The results of this study are reported in Table 4.6. Should we train HydraNet only on the 25% of the current training data, the MRSE becomes 4.5×10^{-2}. Training HydraNet on 75% of the training data helps decreasing the MRSE to 2.6×10^{-2}, implying that adding more training data does help. However, increasing the training data to 100%, does not yield a significant decrease of the MRSE value. It is therefore reasonable to infer that collecting more training data than what we already have would have little impact on further decreasing the MRSE on the test data.

4.7.7. A NOTE ON COMPUTATIONAL COSTS

An objective comparison of the computational costs of FEM and HydraNet analyses is very difficult, if not practically impossible. One should study the efficiency of a HydraNet surrogate on a case-by-case basis, considering the available hardware and implementations. It is nevertheless possible to list some advantages of a CNN-based surrogate implementation over a FEM implementation:

- a CNN-based implementation does not require matrix inversion to obtain the solution array;
- a CNN-based implementation is independent of the constitutive model and the computational cost of its evaluation;
- the most computationally expensive part of HydraNet is a convolution operation which can be performed as a matrix multiplication [19]; and
- a CNN-based implementation can evaluate a batch of specimens in a vectorized manner.

In the examples reported in this section, HydraNet is on average approximately one order of magnitude faster than the corresponding FEM analysis for a given battery cell geometry. We stress that, even if a simulation involves a single time-step (recall the steady state nature of the governing equations, Sec. 4.2), a simulation in a nonlinear finite element framework entails multiple iterations. We do acknowledge that this speed-up is based on our implementations (refer to Sections 4.2 and 4.6) and could significantly vary with another.

4.8. CONCLUDING REMARKS

We proposed a specific type of convolutional neural network (CNN) as an efficient surrogate for the FEM analysis of multi-physics problems. Unlike a FEM analysis, the evaluation of the CNN-based surrogate does not require matrix inversion, and its nonlinear functions are computationally less expensive than those of a FEM analysis.

Previously, in the conclusion section of Ref. [35], we discussed the viability of a surrogate model. An analogous conclusion holds here: there exist a trade-off between the computational time spent on the training of HydraNet (including the data collection) and the computational time saved during a multi-query application. A setting in which the latter is significantly greater than the former is a setting in which HydraNet is an efficient surrogate for the FEM analysis.

A HydraNet can be trained to accurately interpolate, but it is notoriously incapable of extrapolation. Therefore, it is often a good idea to define a certain limit at which predictions of a HydraNet is essentially an interpolation between training data. In this contribution, we defined such a limit through the size and the resolution of the grid. The size of the grid explicitly defines an upper limit to how big a battery cell specimen can be. A specimen larger than the grid simply can not fit into the grid. The resolution of the grid implicitly defines a lower limit to the size of the specimen. If a specimen is so small that only a small number of grid points cover its geometry, the interpolation of the solution field onto the grid becomes extremely inaccurate and, as a direct consequence, HydraNet cannot be accurately trained.

Finally, kernel tensors, bias vectors, and hyperparameters bear no physical meaning. These numerical quantities are tuned based on a dataset and could change dramatically if the dataset is changed.

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, *Gradient-based learning applied to document recognition,* Proceedings of the IEEE **86**, 2278 (1998).

[2] W. Waag, C. Fleischer, and D. U. Sauer, *Critical review of the methods for monitoring of lithium-ion batteries in electric and hybrid vehicles,* Journal of Power Sources **258**, 321 (2014).

[3] L. Wu, X. Fu, and Y. Guan, *Review of the remaining useful life prognostics of vehicle lithium-ion batteries using data-driven methodologies,* Applied Sciences **6**, 166 (2016).

[4] S. Grewal, *A novel technique for modelling the state of charge of lithium ion batteries using artificial neural networks,* in *Twenty-Third International Telecommunications Energy Conference. INTELEC 2001* (IEE, 2001).

[5] X. Dang, L. Yan, K. Xu, X. Wu, H. Jiang, and H. Sun, *Open-circuit voltage-based state of charge estimation of lithium-ion battery using dual neural network fusion battery model,* Electrochimica Acta **188**, 356 (2016).

[6] E. Chemali, P. J. Kollmeyer, M. Preindl, and A. Emadi, *State-of-charge estimation of li-ion batteries using deep neural networks: A machine learning approach,* Journal of Power Sources **400**, 242 (2018).

[7] Y. Zhang, R. Xiong, H. He, and M. G. Pecht, *Long short-term memory recurrent neural network for remaining useful life prediction of lithium-ion batteries,* IEEE Transactions on Vehicular Technology **67**, 5695 (2018).

[8] K. A. Severson, P. M. Attia, N. Jin, N. Perkins, B. Jiang, Z. Yang, M. H. Chen, M. Aykol, P. K. Herring, D. Fraggedakis, M. Z. Bazant, S. J. Harris, W. C. Chueh, and R. D. Braatz, *Data-driven prediction of battery cycle life before capacity degradation,* Nature Energy **4**, 383 (2019).

[9] A. Cecen, H. Dai, Y. C. Yabansu, S. R. Kalidindi, and L. Song, *Material structure-property linkages using three-dimensional convolutional neural networks,* Acta Materialia **146**, 76 (2018).

[10] X. Li, Z. Liu, S. Cui, C. Luo, C. Li, and Z. Zhuang, *Predicting the effective mechanical property of heterogeneous materials by image based modeling and deep learning,* Computer Methods in Applied Mechanics and Engineering **347**, 735 (2019).

[11] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, *Accelerating eulerian fluid simulation with convolutional networks,* in *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (JMLR. org, 2017) pp. 3424–3433.

[12] N. Winovich, K. Ramani, and G. Lin, *Convpde-uq: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains,* Journal of Computational Physics (2019).

[13] Z. Jiang, J. Li, Y. Yang, L. Mu, C. Wei, X. Yu, P. Pianetta, K. Zhao, P. Cloetens, F. Lin, and Y. Liu, *Machine-learning-revealed statistics of the particle-carbon/binder detachment in lithium-ion battery cathodes,* Nature Communications **11** (2020).

[14] H. Sasaki and H. Igarashi, *Topology optimization accelerated by deep learning,* IEEE Transactions on Magnetics **55**, 1 (2019).

[15] A. Khan, V. Ghorbanian, and D. Lowther, *Deep learning for magnetic field estimation,* IEEE Transactions on Magnetics **55**, 1 (2019).

[16] 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 **15**, 20170844 (2018).

[17] D. Grazioli, V. Zadin, D. Brandell, and A. Simone, *Electrochemical-mechanical modeling of solid polymer electrolytes: Stress development and non-uniform electric current density in trench geometry microbatteries,* Electrochimica Acta **296**, 1142 (2019).

[18] D. Grazioli, O. Verners, V. Zadin, D. Brandell, and A. Simone, *Electrochemical-mechanical modeling of solid polymer electrolytes: Impact of mechanical stresses on li-ion battery performance,* Electrochimica Acta **296**, 1122 (2019).

[19] *CS231n: Convolutional Neural Networks for Visual Recognition,* `http://cs231n.github.io/convolutional-networks/`, accessed: 10 October 2020.

[20] G. H. Silva, R. Le Riche, J. Molimard, and A. Vautrin, *Exact and efficient interpolation using finite elements shape functions,* European Journal of Computational Mechanics/Revue Européenne de Mécanique Numérique **18**, 307 (2009).

[21] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,* Nature Methods **17**, 261 (2020).

[22] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient backprop,* in *Neural networks: Tricks of the trade* (Springer, 2012) pp. 9–48.

[23] *Data leakage in machine learning,* `https://machinelearningmastery.com/data-leakage-machine-learning`, accessed: 2019-12-16.

[24] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning* (2020) `https://d2l.ai`.

[25] V. Dumoulin and F. Visin, *A guide to convolution arithmetic for deep learning,* arXiv preprint arXiv:1603.07285 (2016).

[26] A. F. Agarap, *Deep Learning using Rectified Linear Units (ReLU),* arXiv preprint arXiv:1803.08375 (2018).

[27] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift,* (2015), arXiv:1502.03167 [cs.LG] .

[28] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization,* in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, edited by Y. Bengio and Y. LeCun (2015).

[29] *Adam optimization algorithm,* `https://www.youtube.com/watch?v=JXQT_vxqwIs`, accessed: 10 October 2020.

[30] 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.

[31] Joblib Development Team, *Joblib: running python functions as pipeline jobs,* (2020).

[32] C. Molnar, *Interpretable Machine Learning* (2019) `https://christophm.github.io/interpretable-ml-book/`.

[33] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks,* in *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010) pp. 249–256.

[34] C. Sammut and G. I. Webb, eds., *Encyclopedia of Machine Learning and Data Mining*, 2nd ed., Springer Reference (Springer, New York, 2017).

[35] F. Ghavamian and A. Simone, *Accelerating multiscale finite element simulations of history-dependent materials using a recurrent neural network,* Computer Methods in Applied Mechanics and Engineering **357**, 112594 (2019).

5

Conclusion

In this research, we investigate speeding up FEM simulations. We develop three methods to do so. The common denominator of these methods is that an ML model is used in place of (a part of) the FEM model.

The major difference between them is their level of intrusion. The method of Chapter 2 is extremely intrusive. To implement it, one needs to change the very source code of the FEM model. The method of Chapter 3 is less intrusive. The surrogate that we proposed replaces the micro model in a FE^2 scheme. To implement it, one only needs to modify the interface of the micro and macro models. The surrogate proposed in Chapter 4 is the least intrusive of all. The surrogate replaces the entire FEM model and as a result, there is no need to make any changes to the FEM model.

The degree of intrusion of a surrogate model is inversely related to their industrial applicability. In the industry, source codes are either locked to safeguard the intellectual property or modifying them is non-trivial due to their complexity. Hence, we envision methods of Chapters 3 and 4 to have value in the industry while the method of Chapters 2 to remain mostly academic.

Methods proposed in the thesis could enable multi-query applications when the FEM model is computationally expensive. Let us consider a few examples.

A car company wants to assess the reliability of its cars in event of an accident. They can simulate the car crash using a computationally expensive FEM model. But to assess the reliability, they need to conduct a Monte-Carlo simulation in which they need to run the FEM model many times. The computational expense of the FEM model renders the Monte-Carlo simulation computationally intractable. However, using an efficient surrogate model in place of the FEM model could enable the Monte-Carlo simulation.

A virtual reality company is developing haptic technology for its devices. They want to enrich their physics engine by incorporating mechanics. However, mechanical models are numerically solved using FEM models, which are too slow for real-time simulations. Replacing the FEM model with an efficient surrogate could enable real-time simulation while enriching the physics engine with mechanics.

The development of an ML surrogate requires a hefty upfront investment. Firstly, one requires to collect a substantial amount of data to tune the parameters of the ML surrogate. To collect a data point, the computationally expensive FEM model needs to be executed. Secondly, the tuning parameters of the ML surrogate could be a tedious procedure. These beg the question, does the achieved speed up worth the upfront investment? Let us consider an example. Let us say that data collection and parameter tuning take 5 and 2 hours, respectively. And the resulting surrogate is 10 minutes faster than the FEM model. Then the surrogate is only efficient in an application that it is used more than $\frac{5+2 \text{ hours}}{10/60 \text{ hours}} = 42$ times. For that reason, we strongly advise considering the return on investment prior to developing these ML surrogates.

We argue that developing an effective ML model on data collected from experimental laboratories is in vain. We deliver two reasons. First, such data is not readily available, because the data collection procedure is tedious. Let us illustrate that with an example. Let us say we conduct a uniaxial tensile experiment and collect a bunch of stress-strain data points. Then we develop an ML model. This model would fail to grasp the Poisson effect. To enrich the model with the Poisson effect, we have to go back to the laboratory and conduct a different experiment and generate new data. This is not only time-consuming but also expensive. Second, an ML model does not necessarily respect the laws of physics. When developing an ML model on FEM generated data, we have the assurance that the more accurate the ML model is, the more it respects physical laws embedded in the FEM model. But such a guarantee does not apply to ML models developed on experimental data from laboratories. For instance, if devices are not calibrated, the ML model still fits perfectly to the data they generate. However, those data negate the laws of physics.

Acknowledgements

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Grant agreement n° 617972.

To begin with, I am indebted to **Prof. B. Sluys** who kindly facilitated the completion of my Ph.D. study at TU Delft. And I would also like to extend my gratitude to **Prof. A. Simone** for admitting me to this Ph.D. program.

Among my research travels, by far the most fruitful was visiting **Dr. Peherstorfer** at the University of Wisconsin. It was the first time I had access to an expert in the field of model order reduction. In three months, I learned more than I would have ever learned on my own in years.

I believe many of us, working on model order reduction in Europe, are thankful to **Prof. D. Ryckelynck** and **Prof. F. Frtizen** for holding the EUROMECH colloquia. They managed to gather together the small but passionate community of researchers working in this area. It has always been a pleasure attending Bad Herrenalb colloquia.

I am immensely thankful for the vibrant **machine learning online community**. Their openness has democratized learning and applying machine learning. Without the rich material that they generously published online, it would have been much more difficult for me to fill in my machine learning knowledge gap.

Without a doubt, the most exciting time of my Ph.D. was when I gave a series of differential equations lectures to the **CIE4190-2016 class**. I put my best foot forward to deliver an engaging and interesting math lecture to over a hundred civil engineering students. Being one of them once, I knew how difficult of a task it was. Civil engineering students are not particularly fond of differential equations! I only knew I succeeded, when at the last moment of the last lecture, they honored me with a round of applause. Honestly, I have never been more proud in my life. I would like to thank each and every one of them for attending my lectures. They made me realize that I am a teacher at heart and that sooner or later, I should get back to it.

In one of the most difficult period of my study, I sought the guidance of my mentor **Prof. M. Stive**. His much-appreciated advice paved the way for overcoming the challenges I was facing at the time.

By the end of my time at TU Delft, I sought help from **Prof. G. Bertotti**. He sincerely tried his best to help me out. I found him a person who truly cares about students.

I would like to express my gratitude to the **TU Delft Graduate School**. Their courses went a long way toward improving my soft skills. Specifically, I would like to mention **Sandra de Koning**. Even after years, I still use skills I learned in her *teaching fundamentals* and *time management* courses on a daily basis.

I am lucky to be surounded by best colleagues and friends. It was an absolute pleasure to share an office with **Mingzhao Zhuo**. He is far more than an officemate to me; he is a dear friend. Saying that coffee breaks with **Ali Paknahad**, **Mohsen Goudarzi**,

Jafar Amani, **Behrouz Arash**, and **Arman Imani** were entertaining is simply an understatement. Truth be told, talking with them made coming to the university much more appealing. There have been several friends, colleagues, and staffs that made the Ph.D. experience much more pleasing. I would also like to thank **Prashanth Srinivasan**, **Lu Ke**, **Osvalds Verners**, **Davide Grazioli**, **Liting Qiu**, **Tiziano Li Piani**, **Dongyu Liu**, **Luiz A. T. Mororo**, **Willem Nobel**, **Amin Karamnejad**, **Mehdi Musivand Arzanfudi**, **Richard Deker**, **Erik Simons**, **Yaolu Liu**, **Marcello Malagu**, **Noori BniLam**, **Frans van der Meer**, **Frank Everdij**, **Iuri B. C. M. Rocha**, **Anneke Meijer**, **Marianthi Sous**, **Arsha Shiri**, and **Manimaran Pari**.

In the end, I would like to thank **my beloved parents**. Without their selfless support, not much of these would have been possible.

Fariborz Ghavamian
September 2021

Curriculum Vitæ

Fariborz Ghavamian

25-06-1989	Born in Tehran, Iran.

Education

2008–2012	B.Sc. in civil engineering Islamic Azad University of Qazvin, Iran
2012–2014	M.Sc. in civil engineering Delft Univerisity of Technology
2015–2019	Ph.D. in computational mechanics Delft Univerisity of Technology *Thesis:* accelerating finite element analysis using machine learning *Promotor:* Prof. dr. ir. L.J. Sluys *Promotor:* Prof. dr. ir. A. Simone

Occupation

2019–present	Data scientist ING bank

List of Publications

3. F. Ghavamian, D. Grazioli, and A. Simone, "A convolution neural network-based surrogate for finite element analysis of multi-physics problems", *under review*.

2. F. Ghavamian, A. Simone, "Accelerating multiscale finite element simulations of history-dependent materials using a recurrent neural network." *Computer Methods in Applied Mechanics and Engineering* 357 (2019): 112594.
   https://doi.org/10.1016/j.cma.2019.112594

1. F. Ghavamian, P. Tiso, and A. Simone, "POD–DEIM model order reduction for strain-softening viscoplasticity." *Computer Methods in Applied Mechanics and Engineering* 317 (2017): 458-479.
   https://doi.org/10.1016/j.cma.2016.11.025