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

Ghavamian, F.; Simone, A.

DOI
10.1016/j.cma.2019.112594

Publication date
2019

Document Version
Final published version

Published in
Computer Methods in Applied Mechanics and Engineering

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

Available online at www.sciencedirect.com
ScienceDirect

Comput. Methods Appl. Mech. Engrg. 357 (2019) 112594

Computer methods in applied mechanics and engineering

www.elsevier.com/locate/cma

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

F. Ghavamian[a,*], A. Simone[a,b]

[a] Faculty of Civil Engineering and Geosciences, Delft University of Technology, Delft, The Netherlands
[b] Department of Industrial Engineering, University of Padova, Padua, Italy

Received 2 February 2019; received in revised form 10 August 2019; accepted 10 August 2019
Available online 23 August 2019

Abstract

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.

Keywords: Machine learning; Recurrent neural network; Deep learning; Multiscale modeling; Viscoplasticity; Strain softening

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) [1]. 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 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) [2–4] 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 8, this surrogate is capable of dramatically accelerating any FE^2 scheme.

[*] Corresponding author.
E-mail addresses: f.ghavamian@tudelft.nl, fari.ghavamian@gmail.com (F. Ghavamian), a.simone@tudelft.nl, angelo.simone@unipd.it (A. Simone).

https://doi.org/10.1016/j.cma.2019.112594

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. [5] who develop a feed-forward neural network as a surrogate to the damage model introduced by Chaboche [6]. Similarly, Lu et al. [7] introduce a feed-forward neural network as a surrogate for a nonlinear effective electric constitutive law [8]. 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 [9–12] 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. [13] proposed a data-driven framework to link information between multiple scales in a recursive homogenization scheme. Bessa et al. [14] 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 3.1. We then modify an RNN [15] model in Section 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 [16] as quickly described in Section 6.1—the consistent tangent is exact up to the machine precision and, using a deep learning library such as TensorFlow [17], 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 8. Importantly, we discuss how to boost the accuracy of the model through retraining.

2. Problem statement

2.1. Multiscale finite element analysis

To begin with, let us consider an FE^2 scheme. An 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.

2.1.1. Macro model

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

$$\nabla \cdot \boldsymbol{\sigma}_{\mathrm{M}} = \mathbf{0} \quad \text{in} \quad \Omega_{\mathrm{M}}, \tag{1}$$

where ∇ 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 $\Gamma_{\mathrm{M}} = \Gamma_{\mathrm{M}}^{u} \cup \Gamma_{\mathrm{M}}^{t}$ on which Dirichlet (on Γ_{M}^{u}) and Neumann (on Γ_{M}^{t}) boundary conditions are applied.

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

$$(\boldsymbol{\sigma}_{\mathrm{M}}, \hat{\boldsymbol{h}}_{\mathrm{M}}) = \mathscr{M}(\boldsymbol{\varepsilon}_{\mathrm{M}}, \dot{\boldsymbol{\varepsilon}}_{\mathrm{M}}, \boldsymbol{h}_{\mathrm{M}}), \tag{2}$$

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 Eq. (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

$$\boldsymbol{K}_{\mathrm{M}}\,\mathrm{d}\boldsymbol{a}_{\mathrm{M}} = \boldsymbol{f}^{\mathrm{ext}}_{\mathrm{M}} - \boldsymbol{f}^{\mathrm{int}}_{\mathrm{M}}, \qquad \boldsymbol{a}_{\mathrm{M}} \leftarrow \boldsymbol{a}_{\mathrm{M}} + \mathrm{d}\boldsymbol{a}_{\mathrm{M}}. \tag{3}$$

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.

2.1.2. Micro model

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

$$\nabla \cdot \boldsymbol{\sigma}_{\mathrm{m}} = \boldsymbol{0} \quad \text{in} \quad \Omega_{\mathrm{m}}, \tag{4}$$

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 Ref. [18]. The stress is related to the strain through the constitutive relation expressed in rate form

$$\dot{\boldsymbol{\sigma}}_{\mathrm{m}} = \boldsymbol{D}^{\mathrm{e}}\,(\,\dot{\boldsymbol{\varepsilon}}_{\mathrm{m}} - \dot{\boldsymbol{\varepsilon}}^{\mathrm{vp}}_{\mathrm{m}}\,), \tag{5}$$

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 $\boldsymbol{\varepsilon}^{\mathrm{vp}}_{\mathrm{m}}$ is the micro viscoplastic strain tensor.

The micro viscoplastic strain grows at the rate of

$$\dot{\boldsymbol{\varepsilon}}^{\mathrm{vp}}_{\mathrm{m}} = \dot{\lambda}\,\frac{\partial \theta}{\partial \boldsymbol{\sigma}_{\mathrm{m}}} \tag{6}$$

where $\dot{\lambda} = \eta\langle\phi\rangle^{\beta}$ is the rate of plastic multiplier, $\langle\rangle$ 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

$$\sigma_{\mathrm{Y}} = \sigma_{\mathrm{Y}_0}(\,(\,1 + a\,)\,e^{-b\kappa} - a\,e^{-2b\kappa}\,) \tag{7}$$

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 Eq. (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 Eq. (4) is solved using the Newton–Raphson method

$$\boldsymbol{K}_{\mathrm{m}}\,\mathrm{d}\boldsymbol{a}_{\mathrm{m}} = -\boldsymbol{f}^{\mathrm{int}}_{\mathrm{m}}, \qquad \boldsymbol{a}_{\mathrm{m}} \leftarrow \boldsymbol{a}_{\mathrm{m}} + \mathrm{d}\boldsymbol{a}_{\mathrm{m}}. \tag{8}$$

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 micro stiffness matrix, $\boldsymbol{a}_{\mathrm{m}}$ is the micro total displacement vector and $\mathrm{d}\boldsymbol{a}_{\mathrm{m}}$ is its incremental value, $\boldsymbol{f}^{\mathrm{int}}_{\mathrm{m}} = \int_{\Omega_{\mathrm{m}}} \boldsymbol{B}^{\mathrm{T}}_{\mathrm{m}}\,\boldsymbol{\sigma}_{\mathrm{m}}\,\mathrm{d}v$ 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.

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 [19], the macroscopic strain is imposed on the boundary Γ_{m} of the micro model via

$$\boldsymbol{a}_{\mathrm{m}} = \boldsymbol{\varepsilon}_{\mathrm{M}}\,\Delta\boldsymbol{x}, \tag{9}$$

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:

$$\boldsymbol{\sigma}_{\mathrm{M}} = \frac{1}{\Omega_{\mathrm{m}}} \sum_{i=1}^{n} \boldsymbol{f}_{\mathrm{m}}^{i} \otimes \boldsymbol{u}_{\mathrm{m}}^{i}, \tag{10}$$

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 Ref. [20, Box 2].

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 (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 (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. 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 ith sequence by superscript (i), the sth 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 $\boldsymbol{E} = (\boldsymbol{\varepsilon}^{[1]}, \dots, \boldsymbol{\varepsilon}^{[s]} ..., \boldsymbol{\varepsilon}^{[S]})$, the ith strain tensor sequence sample as $\boldsymbol{E}^{(i)}$, and the ith 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 2.1.2 maps a sequence of strain tensors $\boldsymbol{E} = (\boldsymbol{\varepsilon}^{[1]}, \dots, \boldsymbol{\varepsilon}^{[S]})$ to a sequence of stress tensors $\boldsymbol{\Sigma} = (\boldsymbol{\sigma}^{[1]}, \dots, \boldsymbol{\sigma}^{[S]})$:

$$\mathscr{M} : \boldsymbol{E} \rightarrow \boldsymbol{\Sigma}. \tag{11}$$

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

$$\mathscr{D} : \boldsymbol{E} \rightarrow \hat{\boldsymbol{\Sigma}} \tag{12}$$

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

$$e = \|\hat{\boldsymbol{\Sigma}} - \boldsymbol{\Sigma}\|_2 \tag{13}$$

between the predictions of the surrogate $\hat{\boldsymbol{\Sigma}}$ and the micro model outputs $\boldsymbol{\Sigma}$ is small. This process, discussed in Section 4, is referred to as training.

To train the surrogate, Eq. (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.

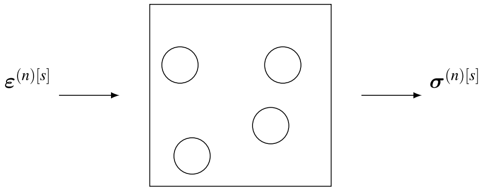


Fig. 1. Collecting strain sequence $\boldsymbol{E}^{(n)}$ and its corresponding strain sequence $\boldsymbol{\Sigma}^{(n)}$ from the standalone micro model. This is a computationally intractable task. The space of all possible strain sequences is extremely large and sampling is computationally intractable.

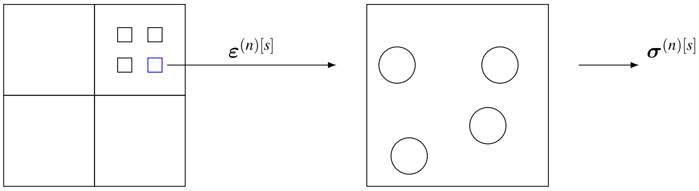


Fig. 2. Collecting strain sequence $\boldsymbol{E}^{(n)}$ and its corresponding strain sequence $\boldsymbol{\Sigma}^{(n)}$. In contrast to the strategy depicted in Fig. 1, here we collect these samples at integration points of an assumed macro model. This is a computationally tractable task. However, the surrogate that is trained on this data set only performs accurately when used in the assumed macro model.

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 Fig. 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 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 Fig. 2 and the data structure is depicted in Table 1.

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 $\boldsymbol{b} = (b^{[1]}, \ldots, b^{[S]})$. Later, in Section 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 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 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.

Table 1
Data collection method: data is collected at each Newton–Raphson iteration.

In nonlinear FE analysis			Sample data from an integration point
Time step: 1	Newton–Raphson iteration: 1	$s = 1$	$\boldsymbol{\varepsilon}^{(n)[1]}$, $\boldsymbol{\sigma}^{(n)[1]}$, $b^{(n)[1]}$
	Newton–Raphson iteration: 2	$s = 2$	$\boldsymbol{\varepsilon}^{(n)[2]}$, $\boldsymbol{\sigma}^{(n)[2]}$, $b^{(n)[2]}$
	Newton–Raphson iteration: 3	$s = 3$	$\boldsymbol{\varepsilon}^{(n)[3]}$, $\boldsymbol{\sigma}^{(n)[3]}$, $b^{(n)[3]}$
Time step: 2	Newton–Raphson iteration: 1	$s = 4$	$\boldsymbol{\varepsilon}^{(n)[4]}$, $\boldsymbol{\sigma}^{(n)[4]}$, $b^{(n)[4]}$
Time step: 3	Newton–Raphson iteration: 1	$s = 5$	$\boldsymbol{\varepsilon}^{(n)[5]}$, $\boldsymbol{\sigma}^{(n)[5]}$, $b^{(n)[5]}$
	Newton–Raphson iteration: 2	$s = 6$	$\boldsymbol{\varepsilon}^{(n)[6]}$, $\boldsymbol{\sigma}^{(n)[6]}$, $b^{(n)[6]}$
			$\boldsymbol{E}^{(n)} = (\boldsymbol{\varepsilon}^{(n)[1]}, \boldsymbol{\varepsilon}^{(n)[2]}, \boldsymbol{\varepsilon}^{(n)[3]}, \boldsymbol{\varepsilon}^{(n)[4]}, \boldsymbol{\varepsilon}^{(n)[5]}, \boldsymbol{\varepsilon}^{(n)[6]})$ $\boldsymbol{\Sigma}^{(n)} = (\boldsymbol{\sigma}^{(n)[1]}, \boldsymbol{\sigma}^{(n)[2]}, \boldsymbol{\sigma}^{(n)[3]}, \boldsymbol{\sigma}^{(n)[4]}, \boldsymbol{\sigma}^{(n)[5]}, \boldsymbol{\sigma}^{(n)[6]})$ $\boldsymbol{b}^{(n)} = (b^{(n)[1]}, b^{(n)[2]}, b^{(n)[3]}, b^{(n)[4]}, b^{(n)[5]}, b^{(n)[6]})$

Table 2
Newton–Raphson iterations of time step t.

Iterations	Previous solution	Current solution
1	a^{t-1}	a_1
2	a_1	a_2
3	a_2	a_3

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 Refs. [21,22]. 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 Fig. 2, we collect samples $\{\boldsymbol{E}, \boldsymbol{\Sigma}, \boldsymbol{b}\}^{(n)}$, where $n = 1, \ldots, N$ and N is the total number of sequences.

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 $\boldsymbol{E} = (\boldsymbol{\varepsilon}^{[1]}, \ldots, \boldsymbol{\varepsilon}^{[S]})$ and convergence indicator sequence $\boldsymbol{b} = (b^{[1]}, \ldots, b^{[S]})$ to stress sequence $\boldsymbol{\Sigma} = (\boldsymbol{\sigma}^{[1]}, \ldots, \boldsymbol{\sigma}^{[S]})$. Different types of RNN models are discussed in Ref. [15]. Here, we propose the RNN model in Fig. 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 ($\boldsymbol{\varepsilon}^{[s]}$, $\boldsymbol{h}^{[s]}$, $\boldsymbol{z}^{[s]}$, $\hat{\boldsymbol{z}}^{[s]}$, $\boldsymbol{\sigma}^{[s]}$).

Let us break down the RNN graph in Fig. 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 3.2.1, 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 $(\boldsymbol{\varepsilon}^{[1]}, \ldots, \boldsymbol{\varepsilon}^{[s-1]})$ 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 [23]. 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 3.2.2. Finally, the dense cell, which is described in Section 3.2.3, maps $\tilde{\boldsymbol{z}}^{[s]}$ to the output stress vector $\hat{\boldsymbol{\sigma}}^{[s]}$.

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.

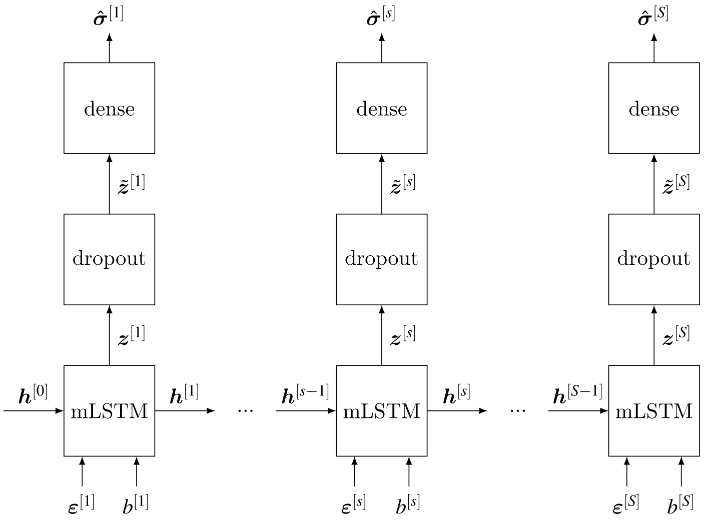


Fig. 3. Recurrent neural network computational graph. It consists of several nodes (mLSTM, dropout, and dense cells) that represent operations and a number of edges that represent operands (the strain vector $\boldsymbol{\varepsilon}^{[s]}$, a set of history vectors $\boldsymbol{h}^{[s]}$, the output of the mLSTM cell $\boldsymbol{z}^{[s]}$, the output of the dropout cell $\hat{\boldsymbol{z}}^{[s]}$, and the output stress vector $\boldsymbol{\sigma}^{[s]}$).

3.2.1. 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. [24,25] for a detailed description). The consequence of vanishing gradients is that the training takes too much time. To overcome this problem, Hochreiter and Schmidhuber [26] suggest employing a specific cell called Long–Short Term Memory (LSTM). A clear explanation of the LSTM cell is given by Olah [27]; 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. [28].

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 are 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

$$\boldsymbol{c}^{[s]} = \boldsymbol{f}^{[s]} \odot \boldsymbol{c}^{[s-1]} + \boldsymbol{i}^{[s]} \odot \hat{\boldsymbol{c}}^{[s]}, \tag{14}$$

then it updates the hidden state

$$\boldsymbol{d}^{[s]} = \boldsymbol{o}^{[s]} \odot \tanh(\boldsymbol{c}^{[s]}), \tag{15}$$

and finally it computes the output vector

$$\boldsymbol{z}^{[s]} = \boldsymbol{o}^{[s]} \odot \tanh(\boldsymbol{c}^{[s]}). \tag{16}$$

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 to zero, we keep using the previous history vectors $\boldsymbol{h}^{[s-1]}$.

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

$$\boldsymbol{i}^{[s]} = \text{sigmoid}(\boldsymbol{W}_e^i \boldsymbol{\varepsilon}^{[s]} + \boldsymbol{W}_d^i \boldsymbol{d}^{[s-1]} + \boldsymbol{b}^i), \tag{17}$$

$$\boldsymbol{f}^{[s]} = \text{sigmoid}(\boldsymbol{W}_e^f \boldsymbol{\varepsilon}^{[s]} + \boldsymbol{W}_d^f \boldsymbol{d}^{[s-1]} + \boldsymbol{b}^f), \tag{18}$$

$$\boldsymbol{o}^{[s]} = \text{sigmoid}(\boldsymbol{W}_e^o \boldsymbol{\varepsilon}^{[s]} + \boldsymbol{W}_d^o \boldsymbol{d}^{[s-1]} + \boldsymbol{b}^o), \tag{19}$$

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

$$\hat{\boldsymbol{c}}^{[s]} = \tanh(\boldsymbol{W}_e^c \boldsymbol{\varepsilon}^{[s]} + \boldsymbol{W}_d^c \boldsymbol{d}^{[s-1]} + \boldsymbol{b}^c), \tag{20}$$

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.

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.

3.2.2. 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 [23].

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

$$\hat{\boldsymbol{z}}^{[s]} = \frac{1}{p}\boldsymbol{r} \odot \boldsymbol{z}^{[s]}, \tag{21}$$

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 regularization—the lower the value of p, the higher the effect of regularization. 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 show that the average value of $\hat{\boldsymbol{z}}^{[s]}$ becomes larger than $\boldsymbol{z}^{[s]}$ [23]. To remedy this issue, the right-hand side of (21) is multiplied by $1/p$ [29].

Training is an iterative procedure and is discussed further in Section 4. 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 [23], an approach that regularizes the model and increase its generalization capabilities.

3.2.3. 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

$$\hat{\boldsymbol{\sigma}}^{[s]} = \boldsymbol{W}\tilde{\boldsymbol{z}}^{[s]} + \boldsymbol{b}, \tag{22}$$

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 $\hat{\boldsymbol{\sigma}}^{[s]}$.

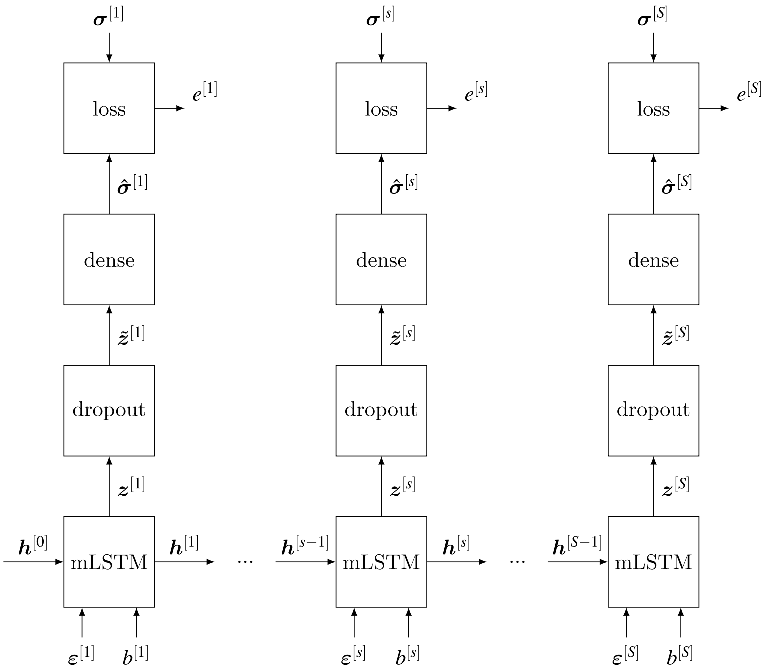

$\boldsymbol{\sigma}^{[1]}$
$\boldsymbol{\sigma}^{[s]}$
$\boldsymbol{\sigma}^{[S]}$
loss
$e^{[1]}$
loss
$e^{[s]}$
loss
$e^{[S]}$
$\hat{\boldsymbol{\sigma}}^{[1]}$
$\hat{\boldsymbol{\sigma}}^{[s]}$
$\hat{\boldsymbol{\sigma}}^{[S]}$
dense
dense
dense
$\tilde{\boldsymbol{z}}^{[1]}$
$\tilde{\boldsymbol{z}}^{[s]}$
$\tilde{\boldsymbol{z}}^{[S]}$
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
$\boldsymbol{\varepsilon}^{[1]}$
$b^{[1]}$
$\boldsymbol{\varepsilon}^{[s]}$
$b^{[s]}$
$\boldsymbol{\varepsilon}^{[S]}$
$b^{[S]}$

Fig. 4. The training graph of the RNN model. The graph in Fig. 4 is essentially the graph in Fig. 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 (23).

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 4.1.

In Sections 3.2.1 and 3.2.3 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 4.2. The training graph of the RNN model is depicted in Fig. 4. The graph in Fig. 4 is essentially the graph in Fig. 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 (23).

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

$$e = \frac{1}{N}\sum_{n=1}^{N} E^{(n)} \quad \text{with} \quad E^{(n)} = \frac{1}{S^{(n)}}\sum_{s=1}^{S^{(n)}} e^{[s]} \quad \text{and} \quad e^{[s]} = \|\boldsymbol{\sigma}^{[s]} - \hat{\boldsymbol{\sigma}}^{[s]}\|_2^2, \tag{23}$$

where N is the total number of training data sets and $S^{(n)}$ is the length of the nth 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.

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 (23) becomes

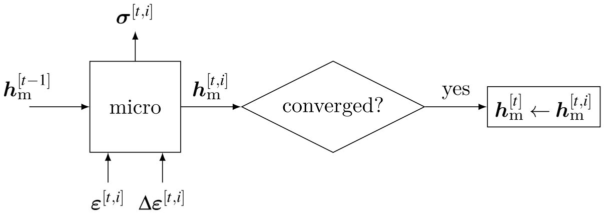

$\boldsymbol{\sigma}^{[t,i]}$
$\boldsymbol{h}_{\mathrm{m}}^{[t-1]}$
micro
$\boldsymbol{h}_{\mathrm{m}}^{[t,i]}$
converged?
yes
$\boldsymbol{h}_{\mathrm{m}}^{[t]} \leftarrow \boldsymbol{h}_{\mathrm{m}}^{[t,i]}$
$\boldsymbol{\varepsilon}^{[t,i]}$
$\Delta\boldsymbol{\varepsilon}^{[t,i]}$

Fig. 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.

small. To this end, we use a specific gradient-based optimizer called Adam [30]. For further reference, Goh [31] explains how optimizers that include momentum, such as Adam, should be understood in an amusingly interactive manner.

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. [30] 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 [32] to tune them.

Due to the bias and variance trade-off [33], 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.

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. Fig. 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 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 Fig. 6. There are a number of differences between the computational graph in Figs. 5 and 6. First, the RNN model does not need the incremental 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.

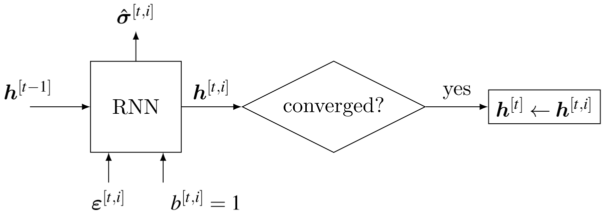

$\hat{\boldsymbol{\sigma}}^{[t,i]}$
$\boldsymbol{h}^{[t-1]}$
RNN
$\boldsymbol{h}^{[t,i]}$
converged?
yes
$\boldsymbol{h}^{[t]} \leftarrow \boldsymbol{h}^{[t,i]}$
$\boldsymbol{\varepsilon}^{[t,i]}$
$b^{[t,i]} = 1$

Fig. 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 3.2.1. 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.

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 [16]. 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 [17] and Pytorch [34], contains an automatic differentiation functionality that can be leveraged to compute the consistent tangent

$$\boldsymbol{D}^{[t,i]} = \frac{\partial \boldsymbol{\sigma}^{[t,i]}}{\partial \boldsymbol{\varepsilon}^{[t,i]}}. \tag{24}$$

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

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 [35]. We implement the RNN model using the TensorFlow library [17].

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.

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 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.

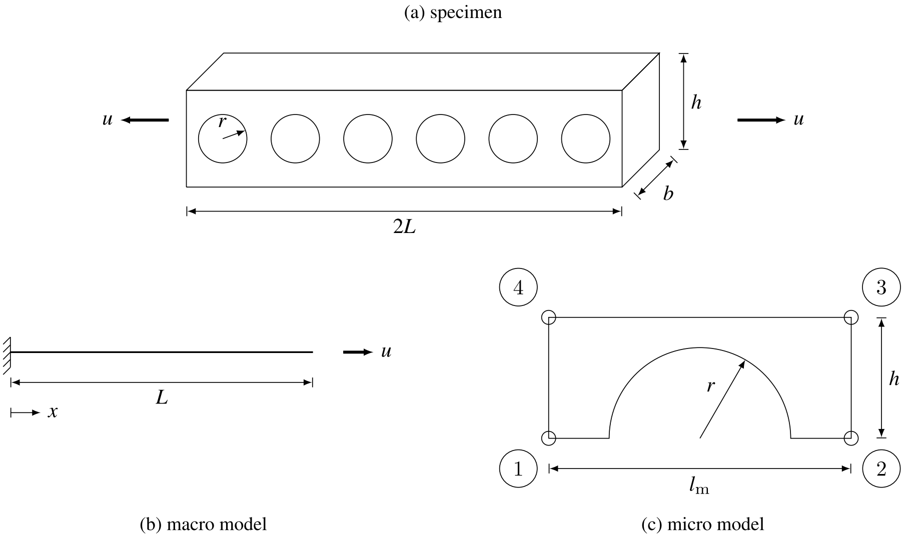


Fig. 7. The length of the specimen (a) is assumed to be significantly larger than its height and width, therefore we model it as a one-dimensional macro model (b) and to incorporate the inclusions we consider the micro model (c).

8. Application

The specimen depicted in Fig. 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 $\sigma_{\text{Y}} = 1$ MPa; viscosity parameter $\eta = 10^{-5}$ s^{-1}. Other model parameters are taken as $\beta = 1$, $a = -1$, and $b = 100$. The use of these parameters is discussed in Section 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 Figs. 7b and 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 Fig. 7c:

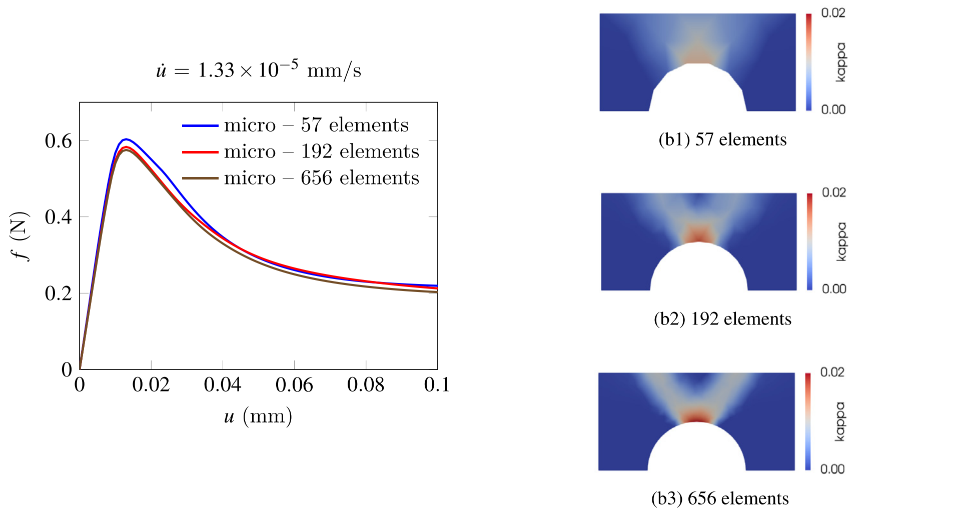
$$\begin{aligned}\boldsymbol{u}_x^{\text{right}} &= \boldsymbol{u}_x^{\text{left}} + \boldsymbol{\varepsilon}\; l_{\text{m}}\\ \boldsymbol{u}_x^{\text{bottom}} &= \boldsymbol{0}\\ u_y^1 = u_x^1 = u_x^4 &= 0\\ u_x^2 = u_x^3 &= \boldsymbol{\varepsilon}\; l_{\text{m}}\end{aligned} \tag{25}$$

Note that the micro model in Fig. 7c does not statistically represent the heterogeneity of the object in Fig. 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 Fig. 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.

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 can be trained to serve as an efficient surrogate for the micro model.



(b1) 57 elements

(b2) 192 elements

(b3) 656 elements

(a) Reaction force f at left boundary of the macro model in Figure 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 8a.

Fig. 8. A uniform mesh refinement study for the micro model. The micro model with 192 triangular elements is selected for the study.

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 Fig. 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 Fig. 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 3.1. 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 3.1, 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 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 Fig. 9. The RNN is not only accurate, but is also considerably faster than the micro model. Based on our implementation, we gain two 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.

8.2. The RNN model for a range of parameters

Let us consider the macro model in Fig. 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;

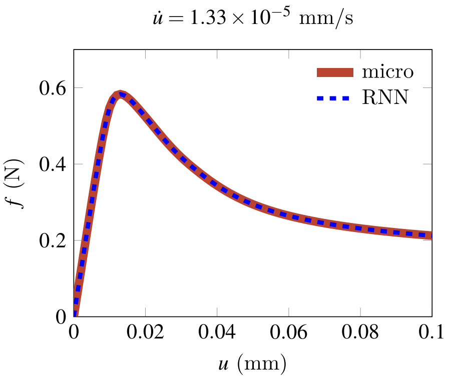


Fig. 9. A sanity check: we train an RNN model on data generated from one configuration of macro model and plot the reaction force f at left boundary of the macro model in Fig. 7b against the imposed displacement u at the right boundary.

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: $u_{\max} \in [0.5, 1.0]$ mm, $\dot{u} \in [1.33, 6.67] \times 10^{-4}$ mm/s. We follow the sampling strategy explained in Section 3.1. For the sake of simplicity, we uniformly sampled the parameters $u_{\max}$ and $\dot{u}$ in the aforementioned ranges. Specifically, we sample 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 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 5. To test the performance of the RNN model, Fig. 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.

Fig. 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 Fig. 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.

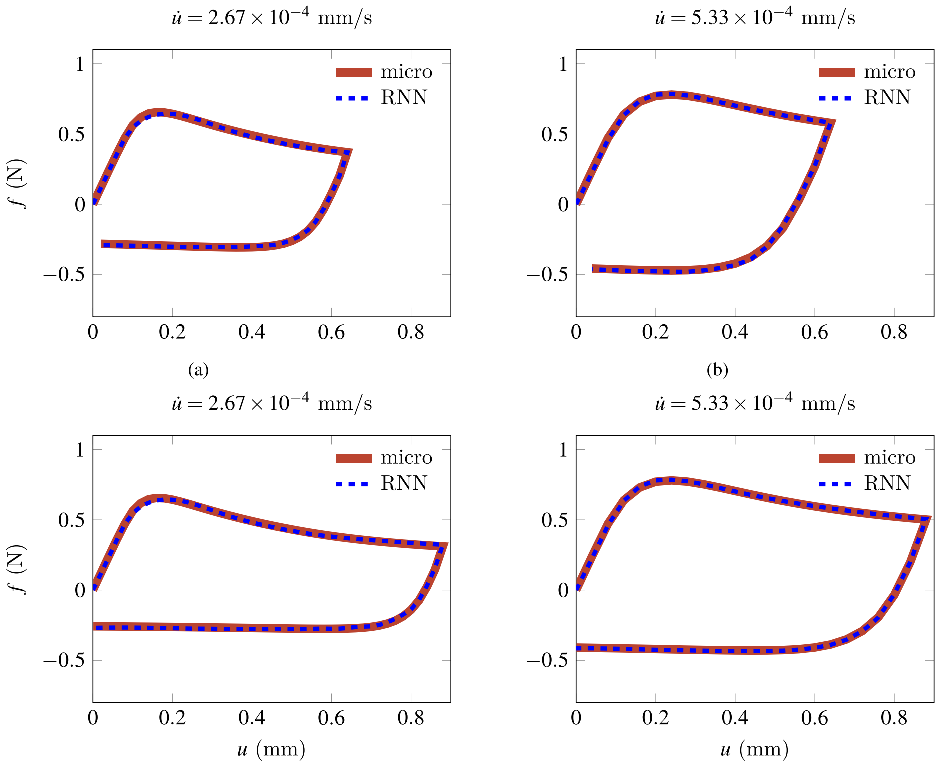
8.3. Retraining the recurrent neural network with new train data

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 $\dot{u} \in \{1.33, 4, 6.67, 13.33\} \times 10^{-4}$ 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 4.2, 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. Fig. 13 shows 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 Fig. 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 [36]. Objectively comparing the impact of the size of the



(a)

(b)

(c)

(d)

Fig. 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 Fig. 7b against the imposed displacement u at the right boundary. None of these four cases was used in the training process.

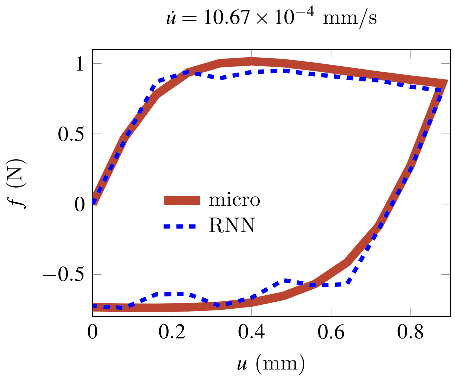


Fig. 11. The RNN model loses accuracy when the imposed displacement rate $\dot{u}$ is out of the range of data the RNN model is trained on, i.e. $\dot{u} \notin [1.33, 6.67] \times 10^{-4}$ mm/s. We show this by plotting the reaction force f at left boundary of the macro model in Fig. 7b against the imposed displacement u at the right boundary.

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 Ref. [37].

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 an FE model in which the strain field localizes.

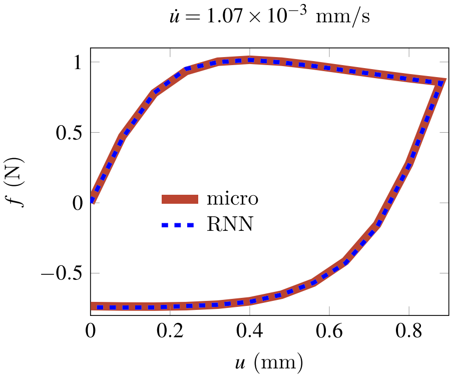
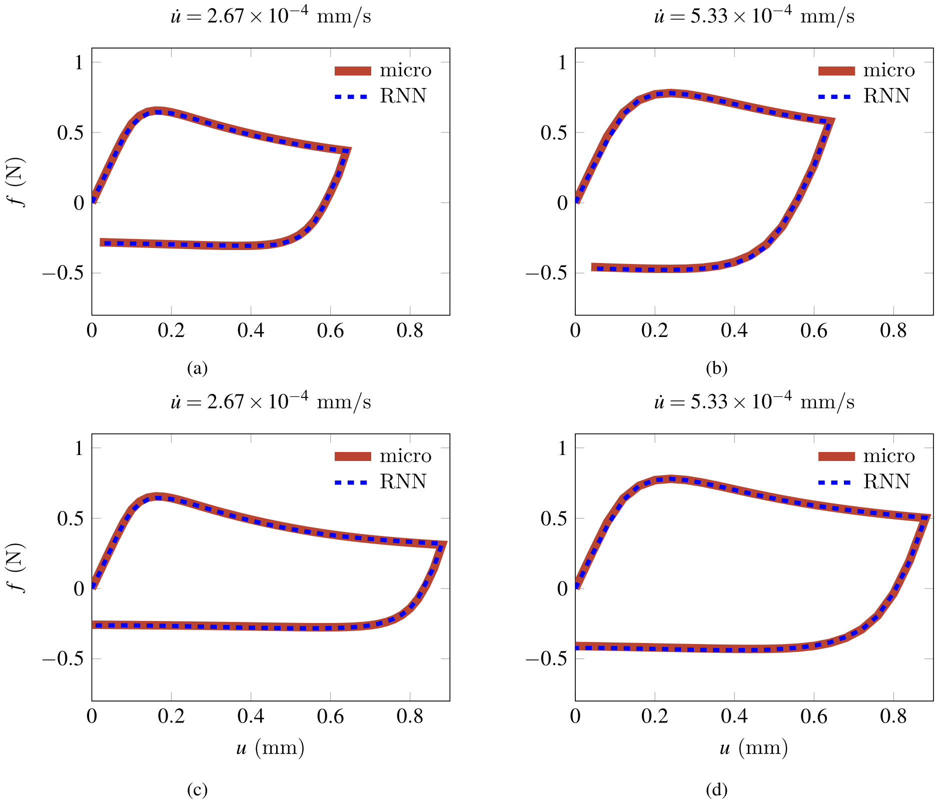


Fig. 12. Comparing this figure with Fig. 11, it is clear that considering an enriched set of data, i.e. $\dot{u} \in \{1.33, 4, 6.67, 13.33\} \times 10^{-4}$ mm/s, boosts the accuracy of the RNN model.



(a)

(b)

(c)

(d)

Fig. 13. The RNN model trained on the enriched set of data performs as accurately as it used to do in Fig. 10.

8.4. Strain localization in a one-dimensional bar

To simulate strain localization, we introduce an imperfection at the center of the bar in Fig. 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, five of which are in the weak zone.

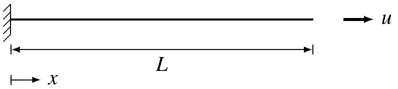
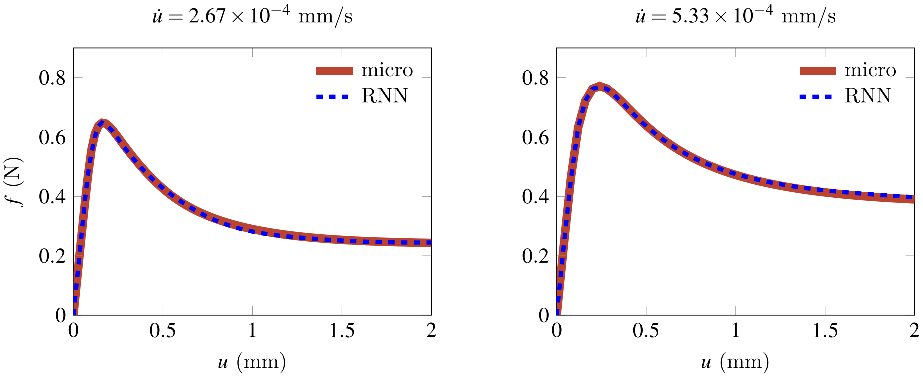


Fig. 14. An imperfection of length $\bar{L} = 0.9L$ is introduced at the center of the macro model in Fig. 7b.



(a)

(b)

Fig. 15. The reaction force f at left boundary of the macro model in Fig. 7b against the imposed displacement u at the right boundary. The average response of the macro model in Fig. 14 appears to be accurate when using the RNN in place of the micro model.

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 multiscale method, to the extent 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 3.1, 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 previous example. We reached these hyperparameters through hyperparameter tuning strategies mentioned in Section 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 Fig. 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 Fig. 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 (Fig. 16a) than when $\dot{u} = 5.33 \times 10^{-4}$ mm/s (Fig. 16b). We also observe that as the strain field localizes further, the discrepancy between the 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 8.3. We do so in the next section.

Let us now study the stress–strain curves, Figs. 17a and 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 becomes larger. In the next section we show that by retraining the RNN model on an enriched set of samples this discrepancy disappears.

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. Specifi-

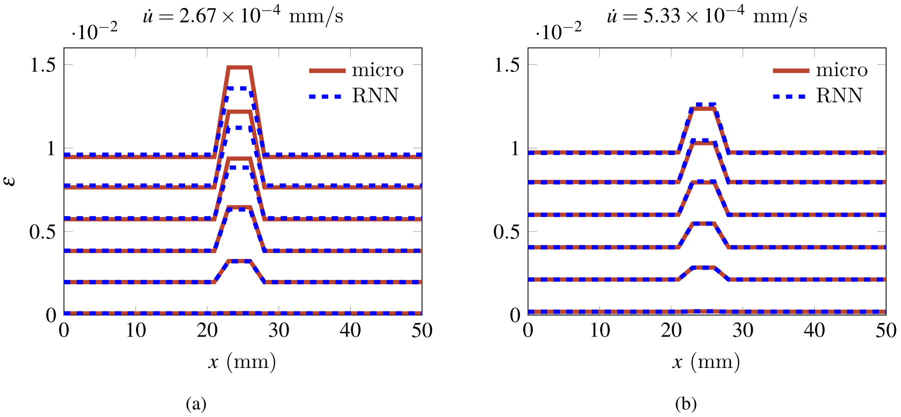


Fig. 16. Evolution of the axial strain field. From Fig. 16a it is apparent that the accuracy of predicted strain field diminishes when the rate of imposed displacement $\dot{u} = 2.67 \times 10^{-4}$ mm/s. In Section 8.4 we remedy this issue by retraining the RNN model on an enriched training set.

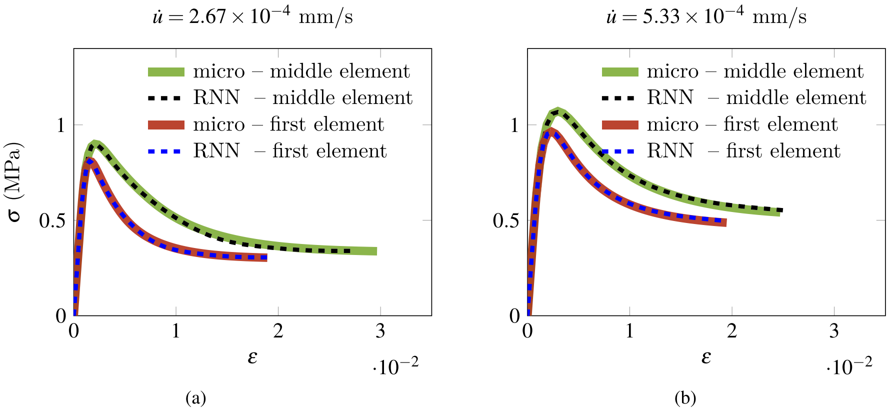


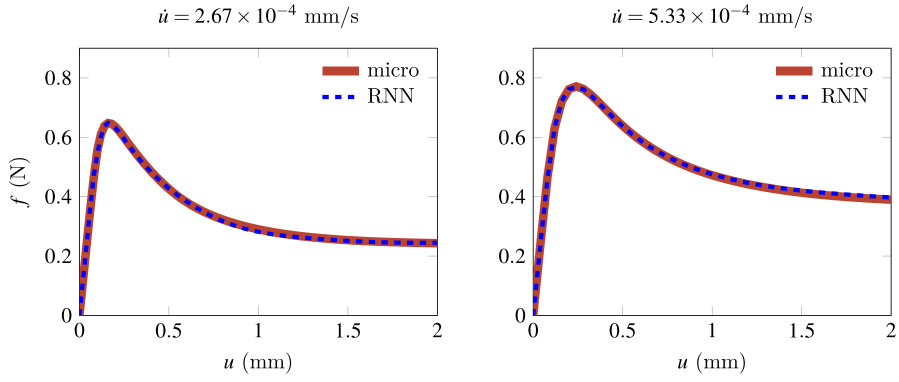
Fig. 17. Strain–strain response of an element in the middle of the macro model, depicted in Fig. 14, and the element adjacent to the clamped boundary.

cally, 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 Fig. 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 Figs. 19a and 16a. The accuracy of the stress–strain curve in Fig. 20a is slightly improved when comparing it with that of in Fig. 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 computational cost of training the RNN model inevitably becomes larger. The computational complexity of evaluating it however remains the same.

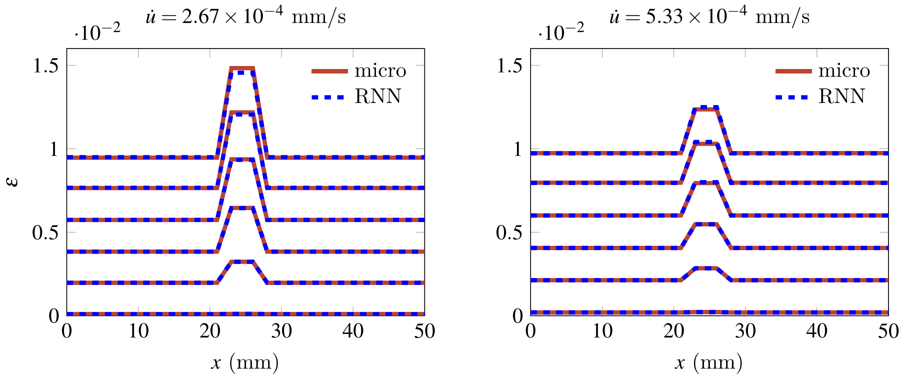
Up until this point we trained two RNN models. One which only works for the macro model with no imperfection and the other for the macro model which exhibits strain localization. In the next section we train one RNN model that works for both these macro models at the same time.



(a)

(b)

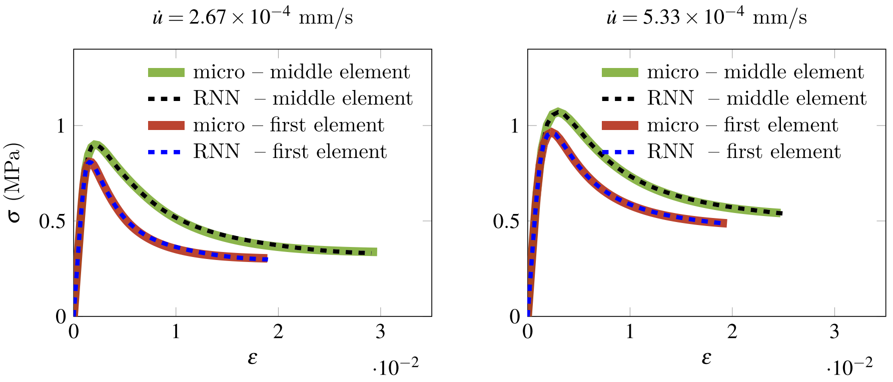
Fig. 18. Reaction force f at left boundary of the macro model in Fig. 14 against the imposed displacement u at the right boundary. Comparing this figure with Fig. 15, it appears that the average response of the system remains accurate when the RNN model is trained on an enriched set of data.



(a)

(b)

Fig. 19. Evolution of the axial strain field. Comparing this figure with Fig. 16 it is clear that, by training the RNN model on an enriched set of data, the accuracy of the predicted strain field increases.



(a)

(b)

Fig. 20. Stress–strain response of an element in the middle of the macro model depicted in Fig. 14 and the element adjacent to the clamped boundary. Comparing Fig. 17a with Fig. 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.

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 Sections 8.3 and 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 Figs. 13a, 13b, 13c, 13d, 12, 15a, and 15b with the newly trained RNN model. We depict the result of these simulations in Figs. 21a, 21b, 21c, 21d, 21e, 22a, and 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 3.1, 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 Figs. 7(b) and 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.

10. Conclusions

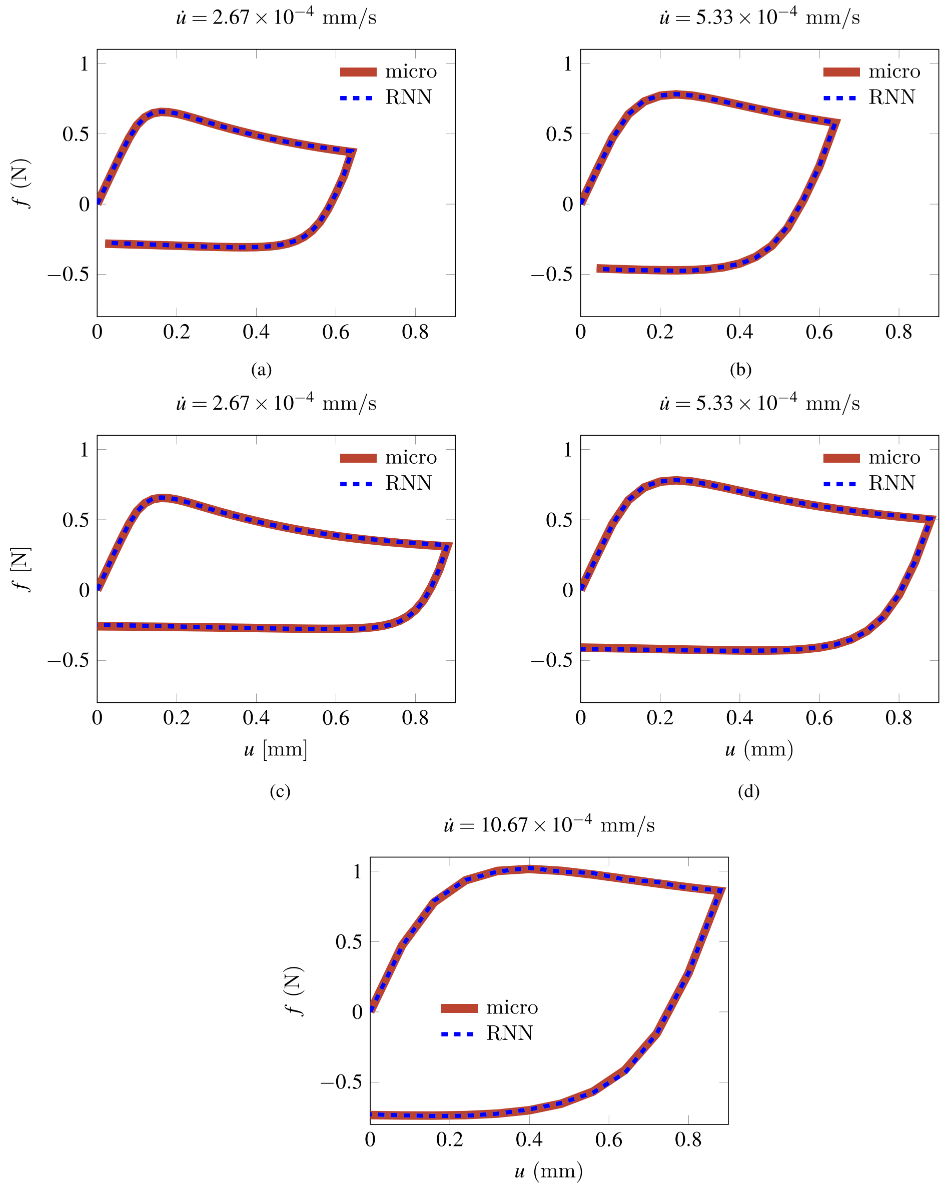
In this contribution we introduced an RNN model that serves as an efficient surrogate for the micro model in a multiscale finite element analysis. We trained the RNN model on a set of data generated by a micro model. The data in the set are collected using an efficient sampling technique that we proposed. 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 h, and it takes 1000 h to train an RNN model. The runtime of the RNN-enhanced multiscale model is only 1 h. 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: $112 \times 10 = 1120$ hours; runtime of the RNN-enhanced multiscale model $1000 + 112 \times 1 = 1112$ 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-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 [38] (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 a 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



(a)

(b)

(c)

(d)

(e)

Fig. 21. Reaction force f at left boundary of the macro model in Fig. 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 Figs. 7b and 14.

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

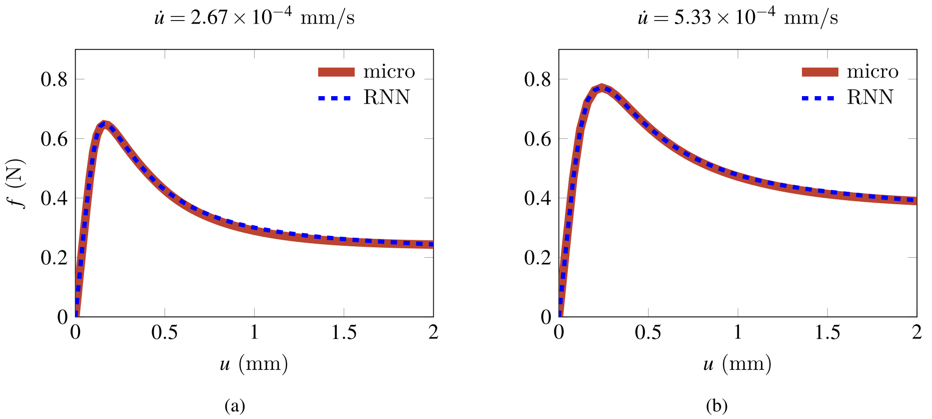


Fig. 22. Reaction force f at left boundary of the macro model in Fig. 14 against the imposed displacement u at the right boundary. The RNN model still performs very accurately when trained on a set of data from both macro models shown in Figs. 7b and 14.

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.

Acknowledgment

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.

References

[1] F. Feyel, A multilevel finite element method (FE^2) to describe the response of highly non-linear structures using generalized continua, Comput. Methods Appl. Mech. Engrg. 28–30 (2003) 3233–3244.

[2] L.C. Jain, L.R. Medsker, Recurrent Neural Networks: Design and Applications, first ed., CRC Press, Inc., Boca Raton, FL, USA, 1999.

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

[4] Z.C. Lipton, J. Berkowitz, C. Elkan, A critical review of recurrent neural networks for sequence learning. arXiv e-prints, art. arXiv:1506.00019, 2015. URL http://arxiv.org/abs/1506.00019v4.

[5] R. Hambli, H. Katerchi, C.L. Benhamou, Multiscale methodology for bone remodelling simulation using coupled finite element and neural network computation, Biomech. Model. Mechanobiol. 10 (2011) 133–145.

[6] J.L. Chaboche, Continuous damage mechanics – A tool to describe phenomena before crack initiation, Nucl. Eng. Des. 64 (1981) 233–247.

[7] X. Lu, D.G. Giovanis, J. Yvonnet, V. Papadopoulos, F. Detrez, J. Bai, A data-driven computational homogenization method based on neural networks for the nonlinear anisotropic electrical response of graphene/polymer nanocomposites, Comput. Mech. 64 (2) (2019) 307–321.

[8] X. Lu, J. Yvonnet, F. Detrez, J. Bai, Multiscale modeling of nonlinear electric conductivity in graphene-reinforced nanocomposites taking into account tunnelling effect, J. Comput. Phys. 337 (2017) 116–131.

[9] L. Xia, P. Breitkopf, A reduced multiscale model for nonlinear structural topology optimization, Comput. Methods Appl. Mech. Engrg. 280 (2014) 117–134.

[10] F. Fritzen, M. Hodapp, The finite element square reduced (FE^{2R}) method with GPU acceleration: towards three-dimensional two-scale simulations, Internat. J. Numer. Methods Engrg. 107 (2016) 853–881.

[11] M. Caicedo, J.L. Mroginski, S. Toro, M. Raschi, A. Huespe, J. Oliver, High performance reduced order modeling techniques based on optimal energy quadrature: Application to geometrically non-linear multiscale inelastic material modeling, Arch. Comput. Methods Eng. (2018) http://dx.doi.org/10.1007/s11831-018-9258-3.

[12] I.B.C.M. Rocha, F.P. van der Meer, S. Raijmaekers, F. Lahuerta, R.P.L. Nijssen, L.P. Mikkelsen, L.J. Sluys, A combined experimental/numerical investigation on hygrothermal aging of fiber-reinforced composites, Eur. J. Mech. A Solids 73 (2019) 407–419.

[13] K. Wang, S. WaiChing, A multiscale multi-permeability poroplasticity model linked by recursive homogenizations and deep learning, Comput. Methods Appl. Mech. Engrg. 334 (2018) 337–380.

[14] M.A. Bessa, R. Bostanabad, Z. Liu, A. Hu, D.W. Apley, C. Brinson, W. Chen, W.K. Liu, A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality, Comput. Methods Appl. Mech. Engrg. 320 (2017) 633–667.

[15] A. Karpathy, The unreasonable effectiveness of recurrent neural networks, 2015. URL http://karpathy.github.io/2015/05/21/rnn-effectiveness/, [Online; accessed January 2019].

[16] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: A survey, J. Mach. Learn. Res. 18 (2018) 1–43.

[17] M. Abadi, A. Agarwal, P. Barham, et al., TensorFlow: Large-scale machine learning on heterogeneous systems 2015. URL https://www.tensorflow.org/, [Online; accessed January 2019].

[18] F. Ghavamian, P. Tiso, A. Simone, POD-DEIM model order reduction for strain-softening viscoplasticity, Comput. Methods Appl. Mech. Engrg. 317 (2017) 458–479.

[19] D. Peric, E.A. de Souza Neto, R.A. Feijoo, M. Partovi, 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, Internat. J. Numer. Methods Engrg. 87 (2010) 149–170.

[20] V.P. Nguyen, O. Lloberas-Valls, M. Stroeven, L.J. Sluys, Computational homogenization for multiscale crack modeling. Implementational and computational aspects, J. Comput. Appl. Math. 234 (2010) 2175–2182.

[21] R. Everson, L. Sirovich, Karhunen-Loève procedure for gappy data, J. Opt. Soc. Amer. A 12 (1995) 1657–1664.

[22] K. Carlberg, C. Bou-Mosleh, C. Farhat, Efficient non-linear model reduction via a least-squares Petrov-Galerkin projection and compressive tensor approximations, Internat. J. Numer. Methods Engrg. 86 (2011) 155–181.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (2014) 1929–1958.

[24] S. Hochreiter, Untersuchungen zu Dynamischen Neuronalen Netzen (Diploma thesis), Technical University of Munich, 1991.

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

[26] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780.

[27] C. Olah, Understanding LSTM Networks, 2015. URL http://colah.github.io/posts/2015-08-Understanding-LSTMs/, [Online; accessed January 2019].

[28] K. Cho, B. van Merrienboer, D. Bahdanau, Y. Bengio, On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. arXiv e-prints, art. arXiv:1409.1259, September 2014.

[29] L. Fei-Fei, J. Johnson, S. Yeung, CS231n Convolutional Neural Networks for Visual Recognition, 2018. URL http://cs231n.github.io/neural-networks-2/, [Online; accessed January 2019].

[30] D.P. Kingma, J. Ba, Adam: A Method for Stochastic Optimization. arXiv e-prints, art. arXiv:1412.6980, December 2014.

[31] G. Goh, Why momentum really works, 2017. URL http://distill.pub/2017/momentum, [Online; accessed January 2019].

[32] J. Snoek, H. Larochelle, 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.

[33] D. Geng, S. Shih, Machine learning crash course: Part 4 - The bias-variance dilemma, 2017. URL https://ml.berkeley.edu/blog/2017/07/13/tutorial-4/, [Online; accessed January 2019].

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

[35] E. Jones, T. Oliphant, P. Peterson, et al., SciPy: Open source scientific tools for Python, 2001. URL http://www.scipy.org/, [Online; accessed January 2019].

[36] A. Ng, Nuts and bolts of applying deep learning, 2016. URL https://www.youtube.com/watch?v=F1ka6a13S9I, [Online; accessed January 2019].

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

[38] A. Khosravi, S. Nahavandi, D. Creighton, A.F. Atiya, Comprehensive review of neural network-based prediction intervals and new advances, IEEE Trans. Neural Netw. 22 (2011) 1341–1356.