Computational Mechanics
https://doi.org/10.1007/s00466-021-01981-y

ORIGINAL PAPER

Recurrent neural networks (RNNs) learn the constitutive law of viscoelasticity

Guang Chen[1]

Received: 25 August 2020 / Accepted: 24 January 2021

Abstract
Recurrent neural networks (RNNs) have demonstrated very impressive performances in learning sequential data, such as in language translation and music generation. Here, we show that the intrinsic computational aspect of RNNs is very similar to that of classical stress update algorithms in modeling history-dependent materials with an emphasis on viscoelasticity. Several numerical examples are designed, including 1-dimensional and 3-dimensional cases, which testify the ability of RNN model to compute the viscoelastic response when predicting on unseen test data. Additionally, it is found that the RNN model trained only on linear and step strain inputs can perform very well on prediction of completely different quadratic strain inputs, demonstrating certain level of generalization ability in extrapolation. Moreover, it is observed that the extrapolation ability depends on the types of strain inputs. The performance is better for continuous strain inputs than that for jump strain inputs. The differences in the generalization ability of RNN models in viscoelasticity and other history-dependent materials are discussed. It suggests that RNN data-driven modeling can be an alternative to the conventional viscoelasticity models.

Keywords Constitutive modeling · Deep learning · History-dependent materials · Recurrent neural networks · Viscoelasticity

1 Introduction

A broad range of engineering materials are history-dependent, with many of them rendering irreversible deformation behaviors and energy dissipation. The constitutive equations describing these materials need to take care of the history associated with material points. For example, the constitutive law of rate-independent J2 plasticity expresses plastic flow rule and hardening law to trace the plastic deformation and yield strength, along with the satisfaction of several additional conditions, such as von Mises yield condition and consistency condition [24,33]. Other examples of history-dependency in constitutive laws include viscoplasticity [29,34], and viscoelasticity [19,20] etc.

The classical mechanical approach to numerically study history-dependent materials is to incorporate the so-called internal state variables (or hidden variables and history variables) that can be applied to track deformation history [18]. This includes many strain-like and stress-like variables, for example the viscous strain or the viscous stress in viscoelastic models. These variables are named so as contrast to the measurable or external variables such as strains and temperatures, and because they carry the history information of the state evolution [18].

While these classic computational mechanics methods have enjoyed great success, the advent of big data and the advances in computer power have supplemented data-driven methods as a new paradigm to the classical computational counterpart [4,21,35]. In particular, the application of the machine learning (ML) technique is becoming more and more popular and accepted. ML and one of its subsets, namely the deep learning (DL) have brought in numerous breakthroughs across various fields, such as material discovery [7,13,28] and drug design [6,8,26,38]. DL takes advantages of deep neural networks (DNNs) and big data to learn the underlying relationships between the inputs and outputs contained in a specific database, such as the material structural information and corresponding material properties in a typical computational mechanics study. However, for a certain type of data that has dependency in the inputs, such as the sequence of sentences in language translation, the traditional DNNs do not perform very well. For this type of problems involving long-term dependency, the recurrent

✉ Guang Chen
guang.chen@uconn.edu

[1] Department of Mechanical Engineering, University of Connecticut, Storrs, CT 06269, USA

Published online: 10 February 2021

neural networks (RNNs) have been shown very effective and successfully applied to problems like natural language processing [9].

This feature of the data type is involved in computational modeling of history-dependent materials, in which the strain and stress sequences are temporally sequential. RNNs have found various applications to model these materials. For example, the first work using RNNs in history-dependent plasticity was by Mozaffar et al. [25]. While there are a number of works using RNNs on plasticity/elastoplasticity modeling [15,25,36] and viscoplasticity modeling [12], few attention is paid to viscoelastic materials. Apparently, this transition would be trivial. But the results show that due to the differences of the nature of viscoelasticity and plasticity, the efforts to train a RNN model and the generalization ability of developed RNNs will be different for viscoelasticity problems and plasticity problems.

The paper is organized as follows. We first draw the similarity of computational facts between the computational modeling of history-dependent materials and the methodology of RNNs following standard presentation in each field. This shall explain why RNN model is naturally adept at this task rather than a brutal-force implementation of DNN-viscoelasticity models. The numerical algorithm for viscoelastic modeling using the generalized Maxwell material model, and RNNs model development are then given in Sect. 3. In Sect. 4, we showcase several numerical examples to demonstrate the performance of the RNNs models. In particular, we test the extrapolation ability of the RNN model on unseen and totally different strain sequences. Finally, we conclude this work with discussions and closing remarks.

2 Intrinsic similarity between the material model of history-dependent materials and RNNs

2.1 Physics-based history-dependent model

In general in numerical computations, the stress update algorithm is employed to keep track of the stresses and internal state variables for history-dependent materials [3]. Using $\boldsymbol{\xi}$ as a collection of state variables, the stress $\boldsymbol{\sigma}$ and state variables $\boldsymbol{\xi}$ at time t_{n+1} can be updated by:

$$\begin{aligned} \boldsymbol{\sigma}_{n+1} &= \boldsymbol{\sigma}(\boldsymbol{\varepsilon}_{n+1}, \boldsymbol{\xi}_n) \\ \boldsymbol{\xi}_{n+1} &= \boldsymbol{\xi}(\boldsymbol{\varepsilon}_{n+1}, \boldsymbol{\xi}_n) \end{aligned} \tag{1}$$

where $\boldsymbol{\varepsilon}_{n+1}$ is the strain measure at time t_{n+1}. Note that equations in Eq. (1) may need to impose other constraints which can be very complex and highly nonlinear, such as the yield condition and plastic flows in elastoplastic and viscoplastic problems. Thus, the classical approach is to use iterative ways, such as Newton-Raphson method, to compute the stresses and state variables. Numerically many algorithms have been successfully applied for these materials, such as return-mapping in plasticity and elastoplasticity problems [30,32]. However, for viscoelastic problems, there are no such additional conditions for admissible stresses. Therefore, it is assumed that the complete stress update algorithm is formulated generally in Eq. (1). We note that this key difference will affect the efforts required in RNN model development and corresponding generalization ability on viscoelasticity problems compared to that on plasticity-related problems.

One can see that stresses at a current time depend on previous state values which recursively rely on more previous states. This characteristic features the essence of history-dependent materials. In this work, we focus only on the viscoelasticity. More details are presented in the later Sections.

2.2 The methodology of RNNs

The general idea of RNN is to use hidden state vectors to store history information in the sequential data. A simple RNN unit (so-called vanilla RNN) is shown in Fig. 1a, in which $a^{\langle t-1\rangle}$ is named the hidden state vector at time t_{n-1}. $x^{\langle t\rangle}$ and $y^{\langle t\rangle}$ are inputs and outputs at time t_n, respectively. The outputs and hidden state vector are updated by:

$$\begin{aligned} a^{\langle t\rangle} &= g(W_{aa}a^{\langle t-1\rangle} + W_{ax}x^{\langle t\rangle} + b_a) \\ y^{\langle t\rangle} &= g(W_{ya}a^{\langle t\rangle} + b_y) \end{aligned} \tag{2}$$

where $g(\cdot)$ is the activation function, such as the sigmoid function. The hyperbolic tangent function $\tanh(\cdot)$ is used for illustration in Fig. 1. W_{aa}, W_{ay}, b_a and b_y are trainable weights and biases to be determined by the training process using the training dataset by minimizing the loss function [14].

Note that there are several different architectures of RNNs, such as one-to-one, one-to-many, many-to-one and many-to-many in terms of input and output sequence [16]. In addition, the length of the inputs and outputs as well as the hidden state vectors can be set flexibly depending on the problem at hand. From the perspective of mechanical modeling, stresses respond to external inputs, *e.g.*, strains or thermal inputs sequentially in time. As a result, the many-to-many architecture is usually adopted. The length of strains and stresses at a single time step can be one or multiple depending on the problem, as will be demonstrated later.

In practical applications, however, the vanilla RNNs have several issues, such as vanishing gradient and not suitable for long-term dependency problems. Advanced units have been more widely applied to address the issues associated with vanilla unit, for example the long short-term memory

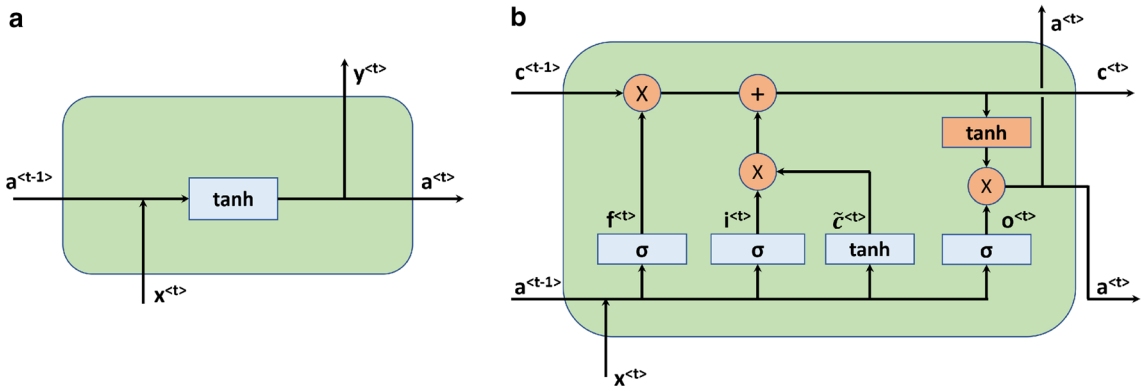

Fig. 1 RNN cell architecture. **a** A simple RNN unit; **b** an LSTM unit

(LSTM) [17] as shown in Fig. 1b and the gated recurrent unit (GRU) [9,36]. In this work, the LSTM is employed and will be introduced later.

One notices that by using the hidden state vector a, the RNN is able to transmit information from earlier states to later states. This intrinsic philosophy exactly resembles the methodology of constitutive modeling of history-dependent materials as presented in Sect. 2.1. This built-in similarity makes the RNNs naturally fit to learn the viscoelastic law. However, an essential question to ask is: is the nonlinear activation function enough to capture the nonlinearity in real materials? Not really, but the universal approximation theorem states that by using several layers of neurons, a neural network can approximate any continuous function [10]. Therefore, RNNs can perform very well to learn the underlying physics-based viscoelasticity law provided that enough and uniform training data is given.

3 Computational methodology

3.1 The generalized Maxwell model for viscoelasticity

Without loss of generality, we consider the infinitesimal-strain viscoelasticity for simplicity of presentation and data generation. Finite-deformation viscoelasticity can be similarly constructed considering the finite-strain kinematics and constitutive law, following *e.g.* the work by Simo and Hughes [31]. Note, however, that in addition to these phenomenological models, there are physics-based models of viscoelasticity [22,23] and viscoelasticity with microstructures [11]. Figure 2 illustrates a 1-dimensional phenomenological model of viscoelasticity, *viz.*, the generalized Maxwell model, which uses springs and dashpots to represent the elastic and viscous response, respectively. This model contains an elastic spring and a number of N Maxwell elements in parallel. The spring element with the modulus μ_∞ is used to represent the ultimate state when the system reaches the equilibrium after a sufficient long time (the rate of change of internal state variables are zeros). The Maxwell elements are used to represent the unsteady responses.

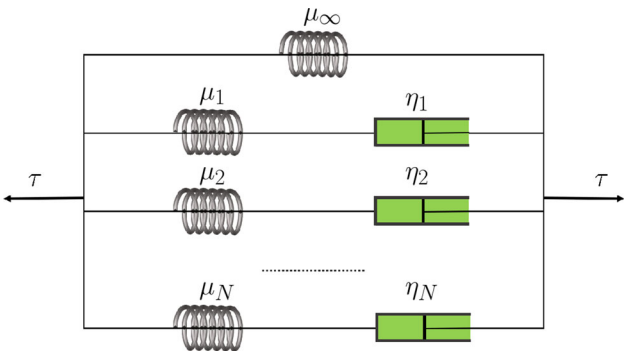

Fig. 2 Phenomenological model of viscoelasticity composed by springs and dashpots to represent elastic and viscous component. The elastic modules and dynamic viscosity are denoted by μ and η, respectively

In addition to the infinitesimal strain assumption, the material is considered to be linear isotropic. Due to the fact that for many viscoelastic materials, such as elastomers, the bulk response is much stiffer than the deviatoric response [31], the material response can be treated separately by two parts: the bulk and deviatoric part. The additive split of the strain is carried out by $\boldsymbol{\varepsilon} = \boldsymbol{e} + \frac{1}{3}\Theta\mathbf{1}$ where $\Theta = \mathrm{tr}[\boldsymbol{\varepsilon}]$ is the volumetric strain and $\boldsymbol{e}$ is the deviatoric strain. In addition, the strain energy density $W^0(\boldsymbol{\varepsilon})$ at initial state (without viscous deformation) is also assumed to be an addition of bulk part $U^0(\Theta)$ and deviatoric part $\bar{W}^0(\boldsymbol{e})$ as $W^0(\boldsymbol{\varepsilon}) = \bar{W}^0(\boldsymbol{e}) + U^0(\Theta)$. Because of the mechanical difference between bulk and deviatoric response, the relaxation in bulk response is neglected.

Introduce the temporal discretization by $[0, T] = \cup[t_n, t_{n+1}]$ and $t_{n+1} = t_n + \Delta t_n$. According to the work by Simo and Hughes [31], the total stress can be computed by:

$$\boldsymbol{\sigma}_{n+1} = U^{0'}(\Theta_{n+1})\mathbf{1} + \gamma_\infty \boldsymbol{S}^0_{n+1} + \sum_{I=1}^{N} \gamma_I \boldsymbol{h}^{(I)}_{n+1} \tag{3}$$

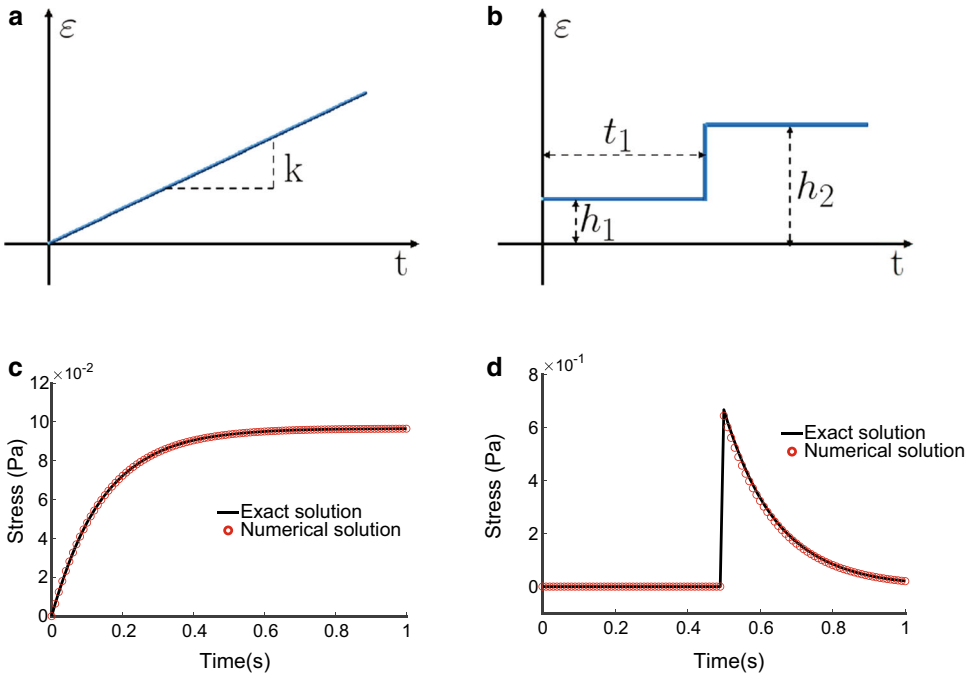

Fig. 3 Strain sequences (ε_{11}) and stress responses under uniaxial loading condition. **a** Linear strain input; **b** step strain input; **c** the stress response (σ_{11}) of linear strain; **d** the stress response (σ_{11}) of step strain. Parameters used: $k = 1 \times 10^{-2}$; $h_1 = 0$; $h_2 = 1 \times 10^{-2}$; $t_1 = 0.5$

where $U^{0'}(\Theta_{n+1})\mathbf{1}$ is the initial bulk stress tensor at time t_{n+1}, $\boldsymbol{S}^0_{n+1}$ the initial elastic deviatoric stress tensor at time t_{n+1}, and $\boldsymbol{h}^{(I)}_{n+1}$ the transient deviatoric stress tensor in the $I-$th Maxwell element at time t_{n+1}. $\gamma_I = \frac{\mu_I}{\mu}$ is the fraction of elastic shear modulus at $I-$th element to that of the total shear modulus $\mu = \mu_\infty + \sum_{I=1}^{N} \mu_I$. The other quantities in Eq. (3) are updated by:

$$\begin{aligned}
\boldsymbol{e}_{n+1} &= \mathrm{dev}[\boldsymbol{\varepsilon}_{n+1}] \\
\boldsymbol{S}^0_{n+1} &= \mathrm{dev}[\partial_{\boldsymbol{e}} \bar{W}^0(\boldsymbol{e}_{n+1})] \\
\boldsymbol{h}^{(I)}_{n+1} &= \exp(-\Delta t_n/\lambda_I)\boldsymbol{h}^{(I)}_n + \exp(-\Delta t_n/2\lambda_I)(\boldsymbol{S}^0_{n+1} - \boldsymbol{S}^0_n)
\end{aligned} \tag{4}$$

where $\lambda_I = \eta_I/\mu_I$ is the relaxation time for I-th Maxwell element. Equations (3) and (4) together define a detailed example of the stress update algorithm for the viscoelastic model corresponding to Eq. (1). Here, $\boldsymbol{h}^{(I)}$ $(I = 1, \ldots, N)$ are the internal stress-like variables, which recursively depend on the values from previous history.

To showcase the stress-strain relation in this viscoelastic model, a single Maxwell element is adopted as the material model with a linear strain and a step strain as the strain inputs as shown in Fig. 3a, b. Unless otherwise stated explicitly, the material parameters in the single Maxwell model used in this work are elastic modulus $E = 100\,\mathrm{Pa}$, Poisson's ratio $\nu = 0.45$ and dynamic viscosity $\eta = 5\,\mathrm{Pa}\cdot\mathrm{s}$. The corresponding stresses for the given strains are shown in Fig. 3c, d. One can see that for the linear strain case, the stress gradually becomes constant, meaning the stress is more maintained by the viscous dashpot as time elapsed while at the very beginning the stress is mainly maintained by the elastic spring (linearly increasing stress). The stress response to the step strain exactly demonstrates the stress relaxation phenomenon (strains are gradually taken up by the viscous dashpot). This viscoelastic behavior is due to the intrinsic nature of elastic springs and viscous dashpots. The former can response instantaneously to external loads while the latter needs time to develop. Thus, there will be stress jumps with step strain inputs.

3.2 Data generation

The database is of great importance in a ML-based study as the ML model relies completely on the data at hand to learn the underlying mechanisms. To that end, two types of strain sequences are considered in the time interval $[0s, 1s]$ with 20 time steps using a single Maxwell model to represent the viscoelasticity. The linear type strain and the step type strain are adopted as shown in Fig. 3. The range of the controlling parameters, *viz.*, the slope k in the linear strain, and the step height h_1, h_2, and the time interval t_1 are set in Table 1.

There are 5000 uniform points sampled for the slope k, and 20 uniform points sampled for the step height h_1, h_2, and the time interval t_1 in the given range listed in Table 1. This gives rise to 5,000 data samples for linear strains and 8000 data samples for step strains. The corresponding stresses can be obtained by the stress update algorithm presented in Sect. 3.1 given the strains generated (the initial stresses are set to be all zeros). To perform this algorithm, the strain tensor is formed by placing the generated 1-dimensional strain data accordingly under the uniaxial loading conditions. In this case, the

Table 1 Parameters setting for strains in data generation

Type	Variable range	No. of uniform points
Linear strain	$k \in [1, 100] \times 10^{-4}$	5000
Step strain	$h_1 \in [0, 10] \times 10^{-3}$	20
	$h_2 \in [0, 10] \times 10^{-3}$	20
	$t_1 \in [0.1, 0.9]$	20

Voigt notation of the strain tensor is $[\varepsilon, -\nu\varepsilon, -\nu\varepsilon, 0, 0, 0]$ considering the Poisson's effect. In generating the database, the material parameters are kept the same as previous settings. Therefore, a database containing 13,000 stress-strain sequences is generated for RNN model development.

3.3 The LSTM unit

The LSTM unit is an advanced version of the vanilla RNN unit, as illustrated in Fig. 1. Compared to the vanilla version, a new state variable $c^{<t>}$ (cell state) is added. In addition, the LSTM unit has three gates (forget gate, input/update gate, and output gate) to determine the information flow in and out of the unit cell. The control algorithm for an LSTM unit is:

$$\begin{aligned} f^{<t>} &= \sigma(W_{fa}a^{<t-1>} + W_{fx}x^{<t>} + b_f) \\ i^{<t>} &= \sigma(W_{ia}a^{<t-1>} + W_{ix}x^{<t>} + b_i) \\ o^{<t>} &= \sigma(W_{oa}a^{<t-1>} + W_{ox}x^{<t>} + b_o) \\ \tilde{c}^{<t>} &= \tanh(W_{ca}a^{<t-1>} + W_{cx}x^{<t>} + b_c) \\ c^{<t>} &= f^{<t>} * c^{<t-1>} + i^{<t>} * \tilde{c}^{<t>} \\ a^{<t>} &= o^{<t>} * \tanh(c^{<t>}) \end{aligned} \tag{5}$$

where $f^{<t>}$, $i^{<t>}$, and $o^{<t>}$ stand for forget, update, and output gate's activated vector, respectively. $\sigma(\cdot)$ is the sigmoid function. $\tilde{c}^{<t>}$ and $c^{<t>}$ are input activated and the state vector. W_{fa}, W_{fx}, W_{ia}, W_{ix},W_{oa}, W_{ox}, W_{ca}, W_{cx}, and b_f, b_i, b_o, b_c are trainable weights and biases. The symbol $*$ denotes the element-wise multiplication.

From the implementation perspective of the RNN model, the input strain $x^{<t>}$ at time t_n can be 1-dimensional (1D), such as $\varepsilon_{11}(t)$ component alone. It can be multi-dimensional, too. For example, the complete strain vector in 3-dimensional (3D) problems using Voigt notation has 6 components. This can be employed easily in a computational code. The code to develop the RNN model is implemented under the platform of Tensorflow package [1].

3.4 RNN model development

The database is split into two sets: a training set with 90% data and a test set with the remaining 10% data since the database has a large amount of data samples. The training set is further divided into a train set and validation set with an 8–2 ratio during training to make sure comparable performances of the model on these two datasets are observed. Namely, a 72%–18%–10% split ratio is applied to the train, validation, and test set, respectively. In the training process, the mean squared error (MSE) between predictions and ground truth values is set as the loss function and the mean absolute error (MAE) is selected as the evaluation metric of the model.

In this work, we systematically test a series of RNN models with different LSTM layers and various number of hidden units (the hyperparameter tuning results can be found in the Supporting Information). For each RNN model, it is trained on the training dataset first. Then the loss and MAE metrics on the unseen test dataset using the RNN model developed in each case can be obtained, which gives guidance on the selection of the best model. For 3D problems, RNN model with two LSTM layers and 50 hidden units for each layer is found to have very good performances. While for 1D problems, RNN model with a single LSTM layer and 5 hidden units is enough to develop very good prediction ability (Results are showed in the Supporting Information).

4 Numerical examples

4.1 Uniaxial example using 3D inputs

For 3D problems, the shape of the strains and stresses in the database are identically [13,000, 120]. In developing the RNN model, this shape has to be reshaped to [13,000, 20, 6] where the 6 indicates the dimension of the strain input at a single time step and there are 20 time steps in total.

The evolution of loss and MAE metrics during the training process on the training and validation dataset is given in Fig. 4a. It can be seen that the performance of the model on both training dataset and validation dataset are in very good agreement, confirming that a stable and robust model has been developed.

After the model is well developed, the prediction ability is then tested using the unseen test dataset. There are 1, 300 strain and respective stress sequences in the test dataset unseen by the developed model, which is then applied to make predictions on this test dataset. Consequently, 1, 300 stress pairs of ground truth and prediction are obtained. A following R^2 correlation analysis for these stress pairs in different components shows that almost all predictions have extremely high correlations close to 1 to the corresponding ground truth, as indicated by Fig. 4b.

To testify the learning ability of the built RNN model, the predicted response and corresponding true viscoelastic response by four randomly selected cases in the test dataset are shown in Fig. 5. Note that since it is an uniaxial tension test, only strain component ε_{22} is plotted to guide the eyes

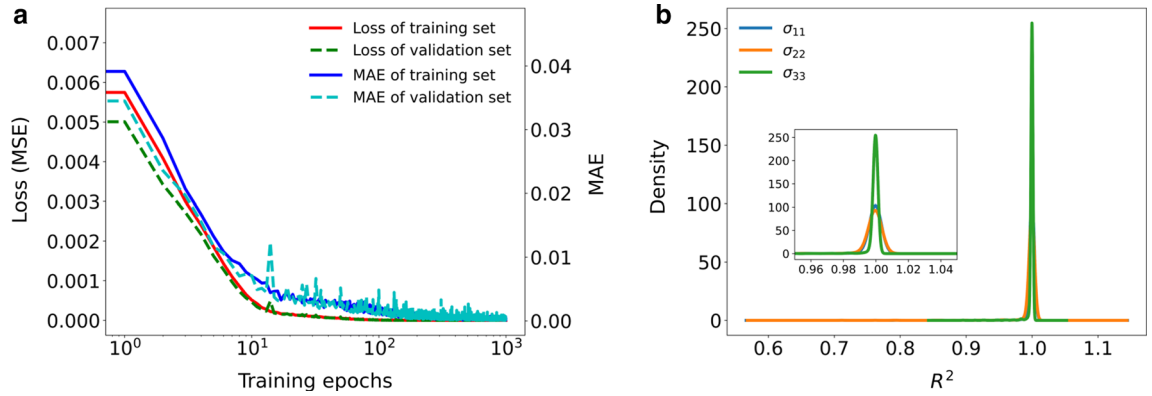

Fig. 4 RNN model evaluation. **a** The loss and MAE evolution of training and validation dataset during training for the RNN model with two RNN layers (each with 50 hidden units); **b** the R^2 correlation score distribution for the predictions of the stresses on the unseen test dataset

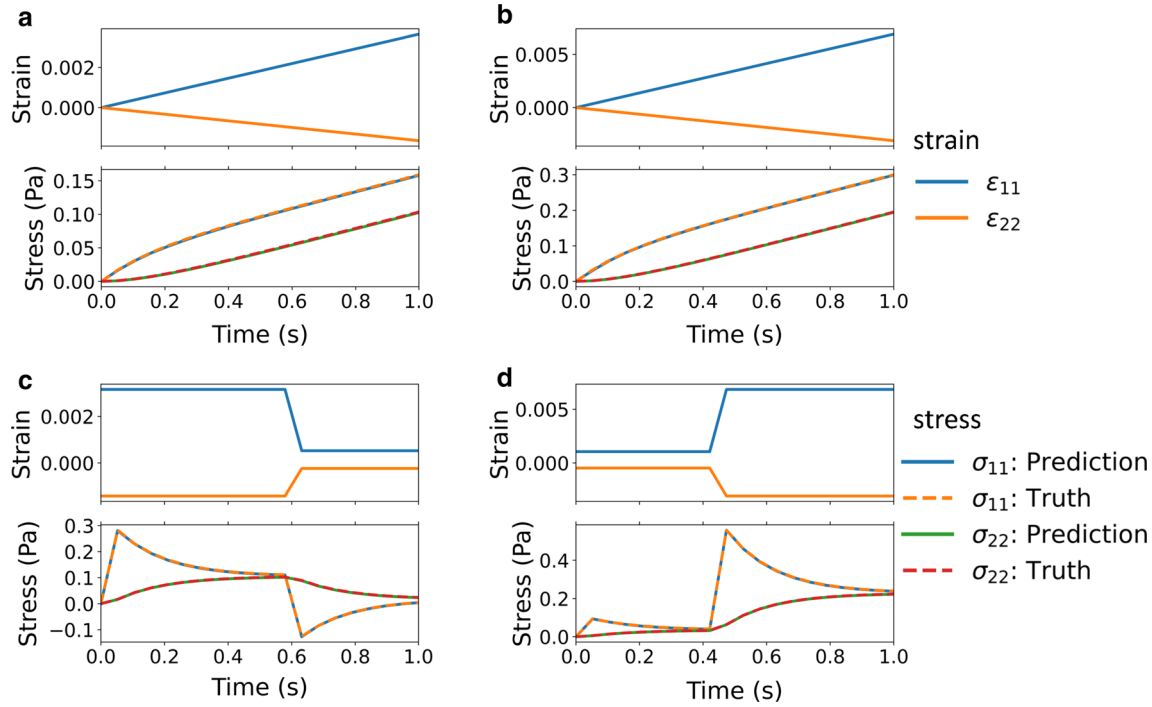

Fig. 5 RNN model evaluation with four randomly selected cases in the test dataset. **a, b** Linear strain cases; **c, d** step strain cases. For each case, the strain sequence is plotted in the above subfigure and the predicted stress as well as the true stress are plotted in the bottom subfigure

as it would otherwise overlap with component ε_{33}. As can be seen from this result that RNNs do perform tremendously well in predicting unseen strain data in both linear strain and step strain cases. Note also that the temporal resolution has an influence on the stress response. A higher resolution than 20 time intervals (used in this work) would be more consistent with theoretical viscoelastic behaviors.

4.2 Learning ability check

In practical situations, a developed computational model is usually applied to external test of similar problems. However, the extrapolation ability of general ML models is not guaranteed since the underlying mechanism for those data beyond the training database may not be consistent with that of the training data. To test whether the developed RNN model is generalizable beyond the training database and to which degree it is generalizable, we further conduct extrapolation tests in two scenarios. In the first scenario, we test the performance of the model on the same strain types (linear and step strain) but with strain ranges beyond the training database; in the second scenario, we test the model on completely different strain sequences, *e.g.* the quadratic stain with $\varepsilon_{11}(t) = \alpha t^2$ and multi-step strain (two steps) sequences.

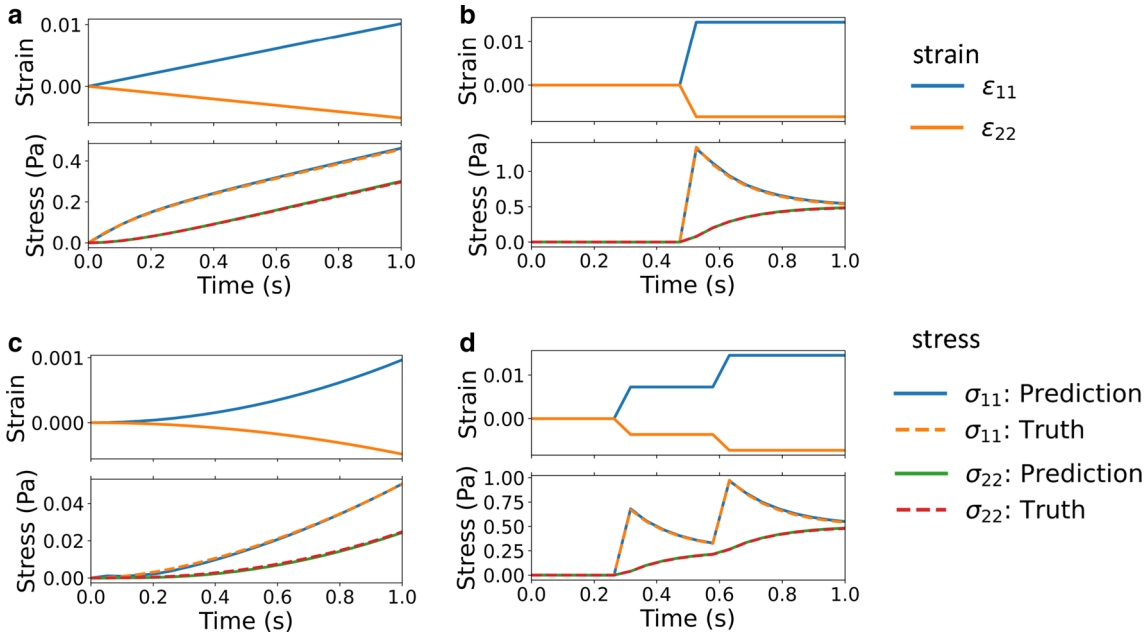

Fig. 6 The extrapolation test (Test-1) of the RNN model using four unseen strain sequences beyond the strains in the training database. For each case, the strain sequence is plotted in the above subfigure and the predicted stress as well as the true stress are plotted in the bottom subfigure

Table 2 Parameters setting for strain generation in the extrapolation tests

Type	Variable value	
	Test-1	Test-2
Linear strain	$k = 105 \times 10^{-4}$	$k = 200 \times 10^{-4}$
Step strain	$h_1 = 0$	$h_1 = 0$
	$h_2 = 15 \times 10^{-3}$	$h_2 = 40 \times 10^{-3}$
	$t_1 = 0.5$	$t_1 = 0.5$
Quadratic strain	$\alpha = 1 \times 10^{-3}$	$\alpha = 10 \times 10^{-3}$
Two-step strain	$h_1 = 0$	$h_1 = 0$
	$h_2 = 7.5 \times 10^{-3}$	$h_2 = 20 \times 10^{-3}$
	$h_3 = 15 \times 10^{-3}$	$h_3 = 40 \times 10^{-3}$
	$t_1 = 0.3$	$t_1 = 0.3$
	$t_2 = 0.6$	$t_2 = 0.6$

These tests are carried out twice using two scales of magnitude, as listed in Table 2.

The developed RNN model in the previous example is again applied to make predictions on these totally new strain sequences. Figure 6 gives the stress-strain plots in the first test (Test-1) with small scale extrapolation, in which the strain sequences are plotted in the upper subfigures. The comparison between the predicted stress responses and the ground truth values are plotted in the corresponding bottom subplots. Surprisingly, all four cases demonstrate good performances in small scale extrapolation, especially for the quadratic and 2-step strain cases since the patterns of them are never seen in the training dataset.

The extrapolation test on the larger scale test (Test-2) is shown in Fig. 7. It is seen that the RNN-viscoelasticity model behaves less well in step strains (single step and multi-step strain) but still performs exceptionally in the linear and quadratic cases. The results show that the RNN-viscoelasticity model behaves better for continuous strain inputs than that for jump strain inputs. As a result, it may be safe to state that proper extrapolation capability of the RNN-viscoelasticity model is demonstrated, upon certain degrees. Therefore, the constructed RNN-viscoelasticity model does learn the physical law of viscoelasticity. But for sake of accuracy, the extrapolation scale should not be too much beyond the training dataset.

4.3 Offline Version of the RNN architecture for high flexibility

Though the developed RNN-viscoelasticity model demonstrates good performances in the above tests, the flexibility of it is limited when applied to external problems (offline applications). This limitation originates from the use of the many-to-many architecture in model training (online training) as shown in Fig. 8a, which uses prescribed strains with fixed length of time steps as the inputs to calculate the stress responses for efficient model training using batches of training data. However, in practical problems, the strains may not be trivial and thus they are not easily prescribed beforehand. Although by padding zeros to the trailing part of the strain inputs, the RNN-viscoelasticity model can still make predictions on the stresses. In this way, the prediction is fea-

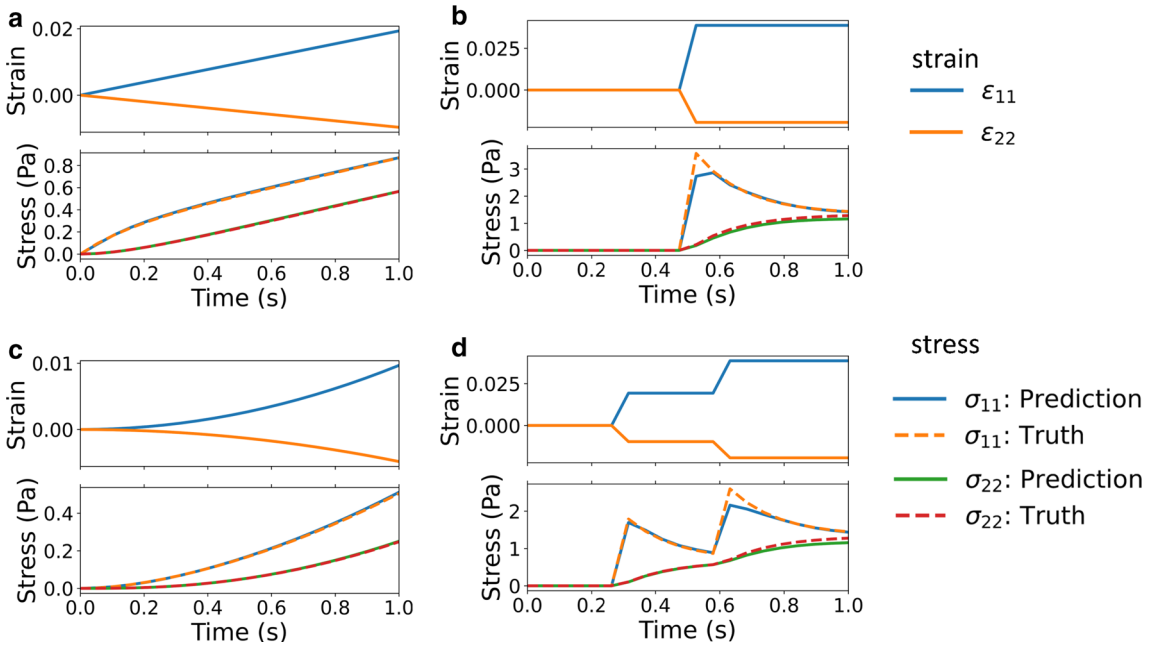

Fig. 7 The extrapolation test (Test-2) of the RNN model using four unseen strain sequences beyond the strains in the training database. For each case, the strain sequence is plotted in the above subfigure and the predicted stress as well as the true stress are plotted in the bottom subfigure

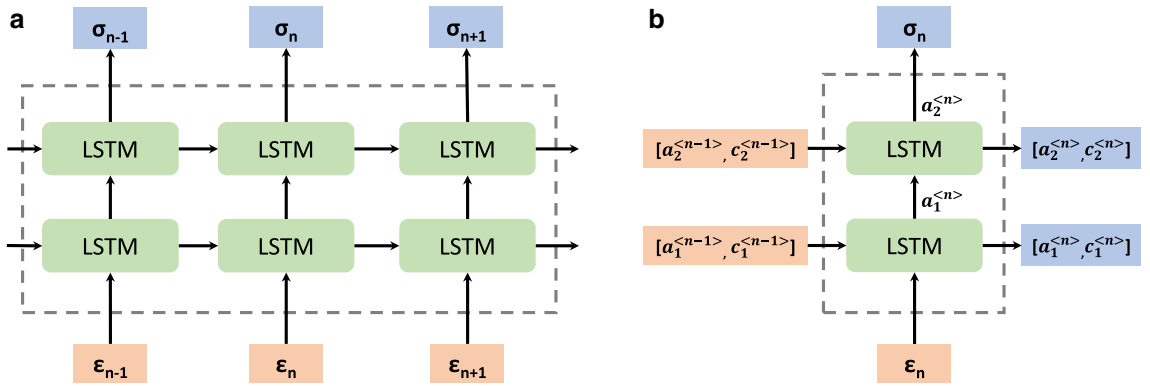

Fig. 8 RNN model architectures. **a** Online training version of many-to-many type for multi-timestep inputs; **b** offline application version of one-to-one type for single timestep inputs. For the state variables, the superscript denotes the time step while the subscript denotes the LSTM layer number

sible, though it is at the cost of unnecessary computation for padded trivial strains at the trailing end. To this end, for offline applications, the architecture can be slightly modified for convenience and efficiency in solving practical problems without any additional computations, as shown in Fig. 8b.

This is inspired by two observations. The first is the classic stress update algorithm, in which the stress responses are updated time wisely with previous state variables as formulated in Eq. (1). The second is that all LSTM cells in the same layer actually share the same weights in the many-to-many architecture and only the hidden states are updating. The realization of the offline version of one-to-one architecture is endowed by the flexibility of the Tensorflow package. The procedure is summarized as follows. After the RNN model is developed, the weights for the LSTM layers and the time-distributed dense layers can be obtained. Then a similar architecture (two LSTM layers plus an additional time-distributed dense layer) with just one time step is created. The weights and biases of the one-to-one architecture is directly assigned without any training using the obtained weights and biases from the well-trained RNN model. Of great importance is that along with the stress output, the updated state variables $[a_1^{<n>}, c_1^{<n>}]$ and $[a_2^{<n>}, c_2^{<n>}]$ in these two LSTM layers will also be returned, which can be easily set in Tensowflow with 'return_state=True'. These updated new states will be used to calculate the next stress responses. At the very beginning, states are usually initialized with zeros (or set in other ways as long as it is consistent with

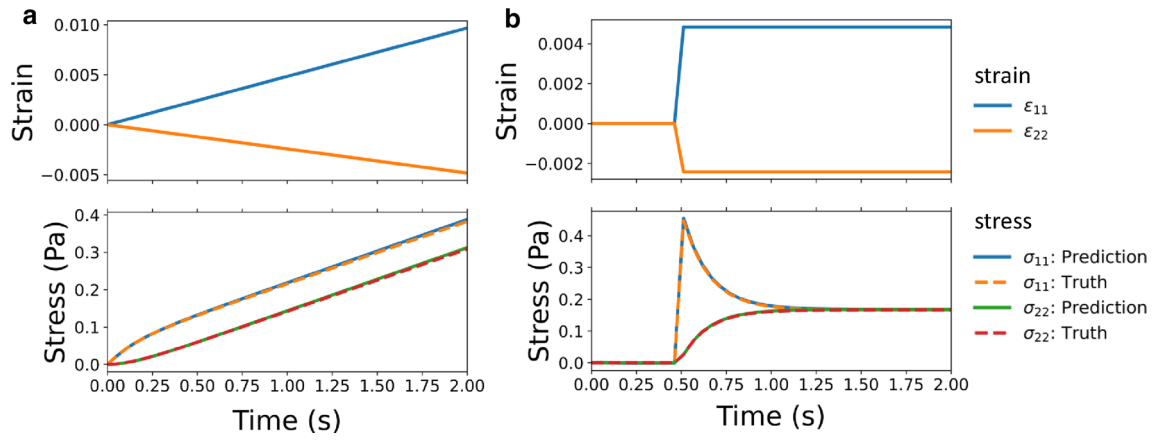

Fig. 9 The prediction results with the one-to-one RNN model. **a** Linear strain; **b** step strain

the settings in training). In this manner, the simplicity of the stress update can be retained as the conventional algorithms.

With this offline version of the RNN model, a linear strain and a step strain case with a longer time are studied. Each case with inputs of 40 time steps (total time of 2 s) as contrast to the 20 time steps in the training dataset (total time of 1 s). The strains are in the same range of magnitude as the training dataset. The predicted responses compared with the true values are plotted in Fig. 9. One can see that very good agreement has been made, justifying the flexibility of the offline version of the RNN architecture for practical problems.

5 Discussion

In general, ML models have difficulties in extrapolation. Depending on the quality of the dataset and the complexity of underlying relationships, the ability of extrapolation can be varied a lot case by case. Generally, it is known that if enough uniform and representative data is given, ML can learn this hidden mapping behind the dataset, however complex they are by the universal approximation theorem [10]. However, in a quantitative sense, it is hard to specify uniform and representative data. As a result, the ability of ML models highly depends on the data at hand. In terms of RNN modeling of history-dependent materials, for example, in a previous work [12], the extrapolation ability of RNN model is very poor in viscoplasticity modeling, while the preliminary results in this work demonstrate pretty good performance to certain degree in extrapolation. We think it is due to the key difference in nature of the constitutive laws in them. For plasticity-related problems, there are more constraints (e.g. plastic flow and hardening law) than viscoelasticity. Additionally, there is no need to deal with admissible stresses bounded by a yield surface in the stress space in the constitutive law of pure viscoelasticity as that of plasticity. Consequently, the efforts to sample uniform and representative data and to train a RNN model are significantly different, which eventually results in differences in the performance of RNN models for different history-dependent materials.

In numerical tests of the RNN-viscoelasticity models, it is also found that the dimensionality of the problems significantly affects the efforts to train the model and its extrapolation ability. When implementing a RNN model with 1D input of strains at each time step, namely the strain sequences of component ε_{11} and stress sequences of component σ_{11} (see in the Supporting Information), use of a single LSTM layer with 5 hidden neurons is enough to have exceptional extrapolation ability even on a larger scale extrapolation. This is a typical characteristic of data-driven studies, which would require trial-and-error and tedious hyperparameter tuning process for complicated problems. Another feature of neural network model is that due to the bias-variance tradeoff, increasing the complexity of the model does not necessarily improve the model performance [12].

For more practical problems involving computational modeling of materials under complex external loading, it is necessary to keep track of the deformation or the displacement of the material. It would require a numerical framework to solve the balance of momentum equation with spatial and temporal discretization (in dynamics modeling), such as finite element method (FEM), together with the constitutive law. This is out of the content of the current work since the integration of FEM and RNNs has actually been studied, for example, in FEM modeling of viscoplastic microstructures [12] and elastoplastic microstructures [36]. Another important step in FEM modeling is to get the consistent tangent to form the stiffness matrix in solving the discretized system equation, which can be feasibly done in coding, attributed to the automatic differentiation technique [12,36] in Tensorflow package. However, if physical conditions are not explicitly enforced, there will be numerical instability issues [37]. For instance, the stiffness matrix should be symmetric and pos-

itive definite, while these conditions may not be satisfied in direct auto-differentiation if lacking explicit enforcement. Using the physics-informed neural network [27] can solve this problem and lead to better generalization ability of the models. This in turn makes the ML model more explainable and generalizable [2,5].

6 Summary and concluding remarks

In this work, the recurrent neural network model was applied to learn the underlying physical law of viscoelasticity. It was motivated by the intrinsic resemblance of the computational aspect of classical mechanical modeling of history-dependent materials and the key idea of RNNs. Numerical examples with 1-dimensional and multi-dimensional linear and step strain inputs were examined under uniaxial loading conditions. The classical viscoelastic modeling scheme serves as a tool to generate the database for RNN models to learn. While in this work only the infinitesimal strain sequences were considered, the extension to the large strain viscoelasticity can also be accommodated following the finite strain viscoelasticity law [31].

The developed RNN models prove notable performances in this study when predicting on the unseen test dataset (linear and step strain inputs), as well as the strain sequences beyond the range of magnitude in the training database to certain degree. It is especially worthy of note in that these constructed RNN models behave very well in generalization investigations to predict on the quadratic strain and multi-step strain sequences at certain level, though the pattern of them is not even present in the training database.

In addition, for offline application using the one-to-one version of the RNN model, the advantages of conventional stress update algorithm can be retained. This architecture is more flexible in predictions of responses in longer time sequences, which is more suitable to solve practical problems.

In summary, in this work the RNN model has been showed to be able to learn the hidden physical law of viscoelasticity pretty effectively. This suggests that deep learning techniques could be very helpful and may become a new paradigm in computational mechanics field.

Acknowledgements The author greatly acknowledges Professor Andrew Ng from Stanford University and his colleagues to make many valuable study resources of machine learning and deep learning freely accessible from the Internet. The valuable comments and suggestions from the editor and anonymous reviewers are greatly acknowledged for improving the quality of the paper.

Data and code availability The data as well as the code for RNNs model development is available upon reasonable request to the corresponding author.

References

1. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, et al (2016) Tensorflow: a system for large-scale machine learning. In: 12th USENIX symposium on operating systems design and implementation (OSDI 16), pp 265–283
2. Adadi A, Berrada M (2018) Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). IEEE Access 6:52138–52160
3. Belytschko T, Liu WK, Moran B, Elkhodary K (2013) Nonlinear finite elements for continua and structures. Wiley, New York
4. Bessa M, Bostanabad R, Liu Z, Hu A, Apley DW, Brinson C, Chen W, Liu WK (2017) A framework for data-driven analysis of materials under uncertainty: countering the curse of dimensionality. Comput Methods Appl Mech Eng 320:633–667
5. Brunton SL, Noack BR, Koumoutsakos P (2020) Machine learning for fluid mechanics. Annu Rev Fluid Mech 52:477–508
6. Burbidge R, Trotter M, Buxton B, Holden S (2001) Drug design by machine learning: support vector machines for pharmaceutical data analysis. Comput Chem 26(1):5–14
7. Chen G, Shen Z, Iyer A, Ghumman UF, Tang S, Bi J, Chen W, Li Y (2020) Machine-learning-assisted de novo design of organic molecules and polymers: opportunities and challenges. Polymers 12(1):163
8. Chen G, Shen Z, Li Y (2020) A machine-learning-assisted study of the permeability of small drug-like molecules across lipid membranes. Phys Chem Chem Phys 22(35):19687–19696
9. Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y(2014) Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv:1406.1078
10. Csáji BC et al (2001) Approximation with artificial neural networks. Fac Sci Etvs Lornd Univ Hung 24(48):7
11. Gao J, Shakoor M, Jinnai H, Kadowaki H, Seta E, Liu WK (2019) An inverse modeling approach for predicting filled rubber performance. Comput Methods Appl Mech Eng 357:112567
12. Ghavamian F, Simone A (2019) Accelerating multiscale finite element simulations of history-dependent materials using a recurrent neural network. Comput Methods Appl Mech Eng 357:112594
13. Gómez-Bombarelli R, Aguilera-Iparraguirre J, Hirzel TD, Duvenaud D, Maclaurin D, Blood-Forsythe MA, Chae HS, Einzinger M, Ha D-G, Wu T et al (2016) Design of efficient molecular organic light-emitting diodes by a high-throughput virtual screening and experimental approach. Nat Mater 15(10):1120–1127
14. Goodfellow I, Bengio Y, Courville A, Bengio Y (2016) Deep learning, vol 1. MIT Press, Cambridge
15. Gorji MB, Mozaffar M, Heidenreich JN, Cao J, Mohr D (2020) On the potential of recurrent neural networks for modeling path dependent plasticity. J Mech Phys Solids 103972
16. Goyal P, Pandey S, Jain K (2018) Deep learning for natural language processing. In: Deep learning for natural language processing: creating neural networks with python. Apress, Berkeley, pp 138–143
17. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
18. Holzapfel AG (2000) Nonlinear solid mechanics II. Wiley, New York
19. Holzapfel GA (1996) On large strain viscoelasticity: continuum formulation and finite element applications to elastomeric structures. Int J Numer Methods Eng 39(22):3903–3926
20. Holzapfel GA, Gasser TC (2001) A viscoelastic model for fiber-reinforced composites at finite strains: continuum basis, computational aspects and applications. Comput Methods Appl Mech Eng 190(34):4379–4403

21. Kirchdoerfer T, Ortiz M (2016) Data-driven computational mechanics. Comput Methods Appl Mech Eng 304:81–101
22. Li Y, Tang S, Kröger M, Liu WK (2016) Molecular simulation guided constitutive modeling on finite strain viscoelasticity of elastomers. J Mech Phys Solids 88:204–226
23. Li Y, Liu Z, Jia Z, Liu WK, Aldousari SM, Hedia HS, Asiri SA (2017) Modular-based multiscale modeling on viscoelasticity of polymer nanocomposites. Comput Mech 59(2):187–201
24. Mises Rv (1913) Mechanik der festen körper im plastisch-deformablen zustand. Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse 1913:582–592
25. Mozaffar M, Bostanabad R, Chen W, Ehmann K, Cao J, Bessa M (2019) Deep learning predicts path-dependent plasticity. Proc Natl Acad Sci USA 116(52):26414–26420
26. Popova M, Isayev O, Tropsha A (2018) Deep reinforcement learning for de novo drug design. Sci Adv 4(7):eaap7885
27. Raissi M, Perdikaris P, Karniadakis GE (2019) Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 378:686–707
28. Ren F, Ward L, Williams T, Laws KJ, Wolverton C, Hattrick-Simpers J, Mehta A (2018) Accelerated discovery of metallic glasses through iteration of machine learning and high-throughput experiments. Sci Adv 4(4):eaaq1566
29. Simo J, Govindjee S (1991) Non-linear b-stability and symmetry preserving return mapping algorithms for plasticity and viscoplasticity. Int J Numer Methods Eng 31(1):151–176
30. Simo J, Hughes T (1987) General return mapping algorithms for rate-independent plasticity. Const Laws Eng Mater Theory Appl 1:221–232
31. Simo J, Hughes TJ (2006) Computational inelasticity, vol 7. Springer, Berlin
32. Simo J, Taylor R (1986) A return mapping algorithm for plane stress elastoplasticity. Int J Numer Methods Eng 22(3):649–670
33. Simo J, Taylor RL (1985) Consistent tangent operators for rate-independent elastoplasticity. Comput Methods Appl Mech Eng 48(1):101–118
34. Simo J, Kennedy J, Govindjee S (1988) Non-smooth multisurface plasticity and viscoplasticity. Loading/unloading conditions and numerical algorithms. Int J Numer Methods Eng 26(10):2161–2185
35. Tang S, Zhang G, Yang H, Li Y, Liu WK, Guo X (2019) Map123: a data-driven approach to use 1d data for 3d nonlinear elastic materials modeling. Comput Methods Appl Mech Eng 357:112587
36. Wu L, Kilingar NG, Noels L et al (2020) A recurrent neural network-accelerated multi-scale model for elasto-plastic heterogeneous materials subjected to random cyclic and non-proportional loading paths. Comput Methods Appl Mech Eng 369:113234
37. Xu K, Huang DZ, Darve E (2020) Learning constitutive relations using symmetric positive definite neural networks. arXiv:2004.00265
38. Zhang L, Tan J, Han D, Zhu H (2017) From machine learning to deep learning: progress in machine intelligence for rational drug discovery. Drug Discov Today 22(11):1680–1685