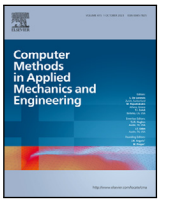
Comput. Methods Appl. Mech. Engrg. 417 (2023) 116432

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma



# Cooperative data-driven modeling

Aleksandr Dekhovich [a], O. Taylan Turan [b], Jiaxiang Yi [a], Miguel A. Bessa [c,*]

[a] Department of Material Science and Engineering, Delft University of Technology, Mekelweg 2, Delft, 2628 CD, The Netherlands
[b] Pattern Recognition and Bioinformatics Laboratory, Delft University of Technology, Van Mourik Broekmanweg 6, Delft, 2628 XE, The Netherlands
[c] School of Engineering, Brown University, 184 Hope St., Providence, RI 02912, USA

ARTICLE INFO

*Keywords:*
Data-driven modeling
Continual learning
Transfer learning
Plasticity

ABSTRACT

Data-driven modeling in mechanics is evolving rapidly based on recent machine learning advances, especially on artificial neural networks. As the field matures, new data and models created by different groups become available, opening possibilities for cooperative modeling. However, artificial neural networks suffer from catastrophic forgetting, i.e. they forget how to perform an old task when trained on a new one. This hinders cooperation because adapting an existing model for a new task affects the performance on a previous task trained by someone else. The authors developed a continual learning method [1] that addresses this issue, applying it here for the first time to solid mechanics. In particular, the method is applied to recurrent neural networks to predict history-dependent plasticity behavior, although it can be used on any other architecture (feedforward, convolutional, etc.) and to predict other phenomena. This work intends to spawn future developments on continual learning that will foster cooperative strategies among the mechanics community to solve increasingly challenging problems. We show that the chosen continual learning strategy can sequentially learn several constitutive laws without forgetting them, using less data to achieve the same error as standard (non-cooperative) training of one law per model.

## 1. Introduction

Machine learning permeated almost every scientific discipline [1–3], and Solid Mechanics is no exception [4–6]. With all their merits and flaws [7], these algorithms provide a means to understand large datasets, finding patterns and modeling behavior where analytical solutions are challenging to obtain or not accurate enough. This work introduces the concept of cooperative data-driven modeling by highlighting the importance of continual or lifelong learning and exemplifying it in Solid Mechanics. Without loss of generality, the examples provided in this article pertain to using neural networks to create constitutive models from synthetic data [4], but the proposed strategy is based on a general method introduced by the authors in the Computer Science community [8], so it is applicable to many other fields that can also benefit from cooperative data-driven modeling.

For readers unfamiliar with the field of using machine learning to learn constitutive models of materials, we provide a short review of the topic. Using neural networks to describe constitutive material behavior was first proposed decades ago by Ghaboussi et al. [9] using simple experimental data. However, advances in numerical modeling and the ability to create large synthetic datasets has led to a new era of data-driven modeling initiated in [4] that is based on fast analysis of representative volume elements of materials. Since then, there has been rapid progress in the field, first by considering similar architectures [10–13], then by considering deep learning strategies to characterize more complex behavior [14–19], and recently including physical

* Corresponding author.
*E-mail address:* miguel_bessa@brown.edu (M.A. Bessa).

[1] The code implementation and data are available at https://github.com/bessagroup/CDDM.

https://doi.org/10.1016/j.cma.2023.116432
Received 1 August 2023; Received in revised form 1 September 2023; Accepted 2 September 2023
Available online 16 September 2023

constraints [20,21]. In particular, the first work to propose the use of recurrent neural networks for plasticity modeling [14] showed that these architectures could learn the path- and time-dependency behavior of materials. Soon after, several research groups proposed new neural network architectures and solved increasingly complex plasticity problems [17,22–24]. A similar trend is ongoing in other fields within and outside Mechanics [25–28].

Simultaneously, the scientific community is experiencing strong incentives to adhere to open science, with vehement support from funding agencies throughout the World to share data and models according to FAIR principles (Findable, Accessible, Interoperable and Reusable) [29–31]. There is also a clear need for end users to reuse these models and data. Nevertheless, there is a serious issue that obstructs the synergistic use of machine learning models by the community. Artificial neural networks, unlike biological neural networks, suffer from catastrophic forgetting [32–34]. Human beings when learning a new task, e.g. playing tennis, do not forget how to perform past tasks, e.g. swimming. Unfortunately, artificial neural networks fail at this because they are based on updating their parameters (weights and biases) for the task and data being considered, but this changes the previous configuration obtained for a past task (that led to different values of weights and biases). This catastrophic forgetting has important implications in practice, as illustrated by the following scenario.

Imagine that Team A of scientists collects computational or experimental data about the constitutive behavior of Material A, and then trains an artificial neural network to predict the behavior of that material. In the end, Team A publishes the artificial neural network model and corresponding data according to FAIR principles. Later, if Team B aims to create a model that predicts the behavior of Material B then it faces two options: (1) collect data and train a model from scratch for this new material; or (2) use the model developed by Team A in an attempt to get a better model for Material B and use less data during the training process. If the material behavior for B has some commonality with the one for A, there is an advantage in leveraging the work from Team A. However, the state of the art in the literature is to use transfer learning or meta-learning methods [35–38] to adapt Model A and retrain it for Team B's scenario [39,40]. Unfortunately, in this case, the new model obtained by Team B is no longer valid for Team A's scenario. Although Team B may create a model that is valid for its purposes, this would not be a truly cooperative effort with Team A because a general model valid for both scenarios would not be obtained. This represents a significant challenge to cooperative data-driven modeling because it discourages different groups from working towards a common model, ultimately leading to many independent models. Note that the problem gets worse as more tasks accumulate (more materials and more teams).

A new branch of machine learning called continual or lifelong learning [41–43] is recently opening new avenues to address the catastrophic forgetting issue. Despite being at an early stage, we believe that addressing this limitation will unlock a new era of cooperative data-driven modeling traversing all fields of application. This year, the authors proposed a new continual learning algorithm [8] and applied it to various standard computer science datasets to demonstrate the best performance to date in the challenging class-incremental learning scenario when compared with state-of-the-art methods. Further developments are needed, but these algorithms represent an essential step towards democratizing cooperative data-driven modeling. Here, the continual learning method is applied to a new architecture suitable for plasticity modeling, demonstrating its benefit and motivating future research in this nascent field.

## 2. Methodology

Continual or lifelong learning [41] aims to learn tasks in a sequence while only having access to the data of the current task. The main challenge in this setting is catastrophic forgetting [32,33], i.e. the phenomenon of forgetting how to solve previous tasks while learning a new one. This problem has become highly significant for deep neural networks in computer vision and natural language processing problems in recent years [42,44–46]. Fundamentally, continual learning tasks are divided into two categories [47,48]: in the first one, the task to be solved during inference is known, i.e. task-ID is given, while in the second it is unknown (task-ID is not given). The former case is more challenging than the latter.[2]

Continual learning algorithms are often classified as regularization, rehearsal or architectural methods [47]. Regularization-based approaches [42,44,49,50] avoid updating weights that are important for predictions of previous tasks. Rehearsal [51–53] methods keep a small portion of the previous data and replay it during training for the new task to prevent forgetting. Currently, rehearsal approaches perform better than regularization-based ones because they keep examples of previous tasks in memory. However, both types of algorithms do not completely prevent forgetting [47], even if the task-ID is given during inference. Therefore, the authors recently developed an architectural method called Continual-Prune-and-Select (CP&S) [8] that exhibits no forgetting when the task-ID is given, and that performs better than state-of-the-art algorithms for different standard computer vision datasets when the task-ID is unknown during inference. Similarly to other architectural methods such as PackNet [54], Piggyback [55], CLNP [56] and SupSup [48], CP&S [8] finds a subnetwork for every task within an original network and uses only one subnetwork for each task. The task-related subnetwork is found by a novel iterative pruning strategy [57], although different methods could be considered [58–60].

Here the focus is to be the first to apply continual learning strategies to mechanics problems, in particular for plasticity modeling. These cases are expected to involve known task-IDs, as illustrated by the previously invoked example of Team A and B that were aiming to model the behavior of two different materials because each team already knows *a priori* the material of interest to them (known task-ID). Therefore, this article only needs to consider the simpler formulation of our CP&S method.

Unlike the standard computer vision problems for which CP&S was originally implemented and compared [8], here we apply CP&S to model irreversible material behavior. Without loss of generality, we focus on plasticity simulations but other history-

[2] For readers unfamiliar with the concept of not knowing the task-ID, think about the previous example of one of the tasks being 'swimming' and the other being 'playing tennis'. If the task-ID is not known during inference, then understanding that the task being executed is 'playing tennis' has to happen from context (from the data). Otherwise, if the task-ID is given there is no need to infer from context which task is being considered.

and time-dependent phenomena such as damage or visco-elasticity could be considered. In fact, CP&S can be applied to other classification or regression tasks (they do not need to be tasks in Mechanics). Given the time and/or history-dependency of these problems, considering neural network architectures with recurrent units facilitates the learning process.

Recurrent neural networks (RNNs) [61,62] typically deal with sequential data, e.g. in natural language processing problems [63] or voice recognition [64]. However, they suffer from vanishing and exploding gradients issues [65,66]. Further improvements of RNNs, such as the Long Short-Term Memory (LSTM) network [67] and Gated Recurrent Unit (GRU) [68] solved this problem, enabling their application in different contexts. In particular, LSTMs and GRUs have been shown to learn history-dependent phenomena in mechanics [14,69]. In this work, we use GRUs as described in our past work [14] due to their simplicity and effectiveness when compared to LSTM, although other neural network architectures with recurrent units could be considered. Computations that occur in the GRU can be described as follows:

$$z_t = \text{sigmoid}\big(W_z x_t + U_z h_{t-1} + b_z\big), \tag{1}$$

$$r_t = \text{sigmoid}\big(W_r x_t + U_r h_{t-1} + b_r\big), \tag{2}$$

$$\hat{h}_t = \tanh\big(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h\big), \tag{3}$$

$$h_t = z_t \odot \hat{h}_t + (1 - z_t) \odot h_{t-1}, \tag{4}$$

where $h_0 = 0$, $x_t$ is an input vector, $t = 0, 1, \dots, T$. Matrices $W_z$, $W_r$, $W_h$, $U_z$, $U_r$, $U_h$ are learnable weights and $b_z, b_r, b_h$ are learnable biases. In this work, the PyTorch [70] implementation of GRUs is used to conduct the experiments consisting of simplified representative volume elements of materials with different microstructures, as described in Section 3.

Independently of the architecture chosen for the neural network, we advocate here that architectural continual learning algorithms such as CP&S [8] enable Cooperative Data-Driven Modeling (CDDM). The method creates different subnetworks within an artificial neural network that are associated to particular tasks without forgetting past tasks. For example, in our context each subnetwork is associated to a specific material model for a class of materials with a given microstructure, constituent properties and external conditions (see e.g. [4]). Architectural methods, however, are limited by the availability of free neural connections that can be trained for a new task. A particular advantage of CP&S is that it is based on the NNrelief iterative pruning strategy [57] which was developed aiming to create sparser subnetworks (using fewer connections) than other pruning methods such as magnitude pruning [59] or neural pruning [60]. Additionally, although CP&S was implemented in the original article for convolutional and fully connected networks, it can easily be adapted to GRU networks as shown in this work.

CP&S is based on a set of simple steps to create a group of overlapping subnetworks, each of them learning a particular task without disrupting the knowledge accumulated by the other subnetworks. Importantly, the subnewtorks can (and usually do) share knowledge among them by sharing connections that are useful to each other. This mechanism allows to learn different tasks, and transfer knowledge between them but avoids forgetting, unlike transfer learning methods.

Overall, cooperative data-driven modeling via the proposed CP&S method is described as follows:

- Training:

  1. At the beginning of the learning process, initialize the entire neural network with random weights (see Fig. 1a).
  2. Set the hyperparameters (architecture, optimizer, learning rate, weight decay, pruning parameters, etc.).
  3. For a task $T_i$ (e.g. learning a new material), create a subnetwork $\mathcal{N}_i$ that is associated to that task:

     (a) Use the NNrelief [57] algorithm to prune connections from the entire neural network (see Fig. 1b, details in Appendix).

     - Remark 1: Every connection can be pruned, whether they are already part of another subnetwork or not.
     - Remark 2: Pruning is controlled by hyperparameter $\alpha \in (0, 1)$ that represents the amount of information that is maintained coming out of the neurons after removing the connections. So, $\alpha = 0.95$ indicates that 95% of the signal coming out of the neurons is kept on average for the training set of task $T_i$ after removing the connections. The amount of information is measured according to a metric called *importance score* – see the original article [57] or Appendix for details.
     - Remark 3: Allowing to prune any connection of the entire network (whether belonging to a previous subnetwork or not) is crucial because it provides a mechanism to keep connections that are useful for performing the current task but remove the ones that are not. If some connections are kept from other subnetworks, then there is potential for knowledge transfer.

     (b) Among the remaining connections, i.e. the ones that are not pruned, retrain the "free" connections but do not update the ones that are part of other subnetworks (see Fig. 1c).

     - Remark 4: By refraining from updating connections that are part of other subnetworks, the CP&S algorithm avoids forgetting past tasks because the other subnetworks are not affected by the current training of this subnetwork. Instead, only connections that have not been used previously by any subnetwork are updated so that new knowledge can be accumulated, as needed to solve a new task.

     (c) Assess performance of this subnetwork ($\mathcal{N}_i$) on the dataset for this task ($T_i$). If error is not acceptable, go to 3(a) and iterate again until a maximum number of iterations $n$ is met. Else save the connections that form this new

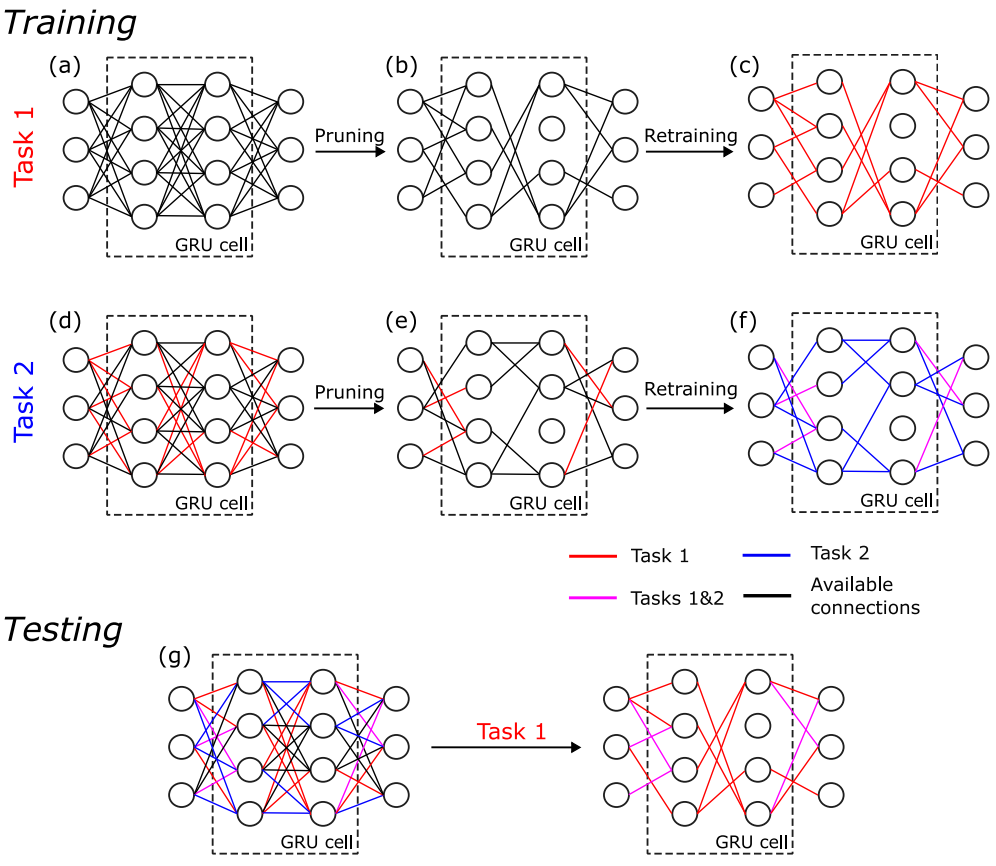



**Fig. 1.** Overview of the proposed CDDM approach.

subnetwork into mask $\mathbf{M}_i$. The weights belonging to this subnetwork will no longer be updated when training another subnetwork later.

4. Consider the next task $T_{i+1}$ by finding a new subnetwork ($\mathcal{N}_{i+1}$) as described in step 3 (see Fig. 1(d–f)).

- Inference (testing):

1. For the given test point $x$ from task $T_i$, select subnetwork $\mathcal{N}_i$ (see Fig. 1g).

   – Remark 5: in this paper, task selection from the data is not necessary, because practitioners already know what is the material model that they want to consider. Otherwise, see the original article for the task incremental learning scenario [8].

2. Make a prediction $y = \mathcal{N}_i(x)$.

We highlight from the above description that the performance of CP&S is controlled only by two hyperparameters: the number of pruning iterations $n$ and the pruning parameter $\alpha \in (0, 1)$. A low value of $\alpha$ will cause more connections to be pruned (information compression) but also lower expressivity (worst performance of the subnetwork after pruning and retraining for $n$ iterations). The subnetwork is represented by a binary mask $\mathbf{M}$, where every active connection is represented by 1 and where every inactive connection is represented by 0. The weights and biases that are first assigned to a particular task are not updated for subsequent tasks – ensuring that the original subnetwork for which these connections were trained remains unchanged by the training process of a new subnetwork. This strategy was proven effective in the context of computer vision, and it should remain valid for computational mechanics applications.

An important disadvantage of CP&S is that fixing the parameters associated with a trained subnetwork can quickly exhaust the number of "free" connections available to be trained in subsequent tasks. In other words, the artificial neural network can be "saturated" after several tasks have been learned. This is reported in the original investigation [8]. However, we note that CP&S does not have limitations on the type of neural network architecture to be considered. Therefore, the choice of architecture depends only on the solvable problem. In this paper, we consider the case when all tasks have the same input and output dimensions. This

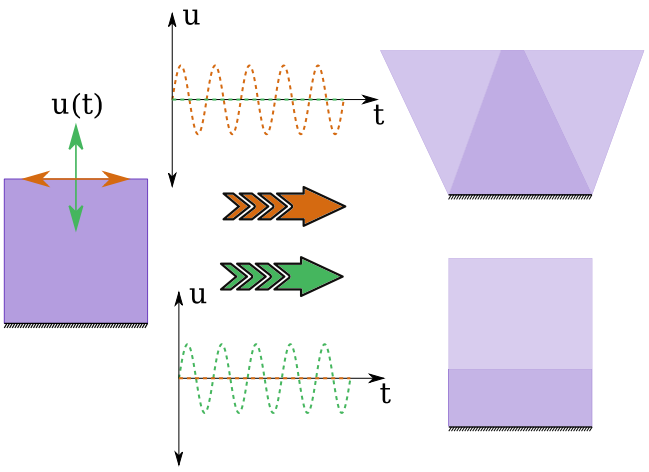
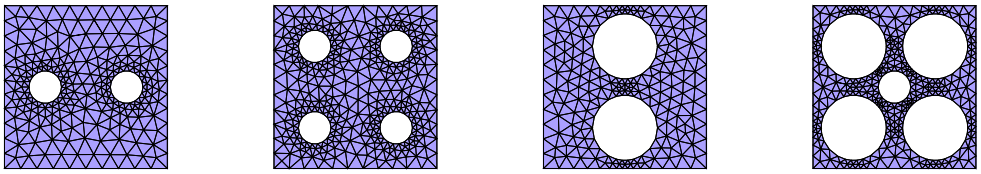



**Fig. 2.** A square domain fixed on the bottom and displaced on the top. Displacement is done in a pseudo-time.



(a) Task A. (b) Task B. (c) Task C. (d) Task D.

**Fig. 3.** Different domains introduced as different tasks.

condition can be relaxed if one uses separate task-specific input and output heads. In this case, shared parameters are the ones that are in the intermediate layers.

## 3. First case study: learning plasticity laws for different microstructures

The first and simpler case study considers non-periodic material domains subjected to uniform displacements at the boundary. The goal is to create constitutive models from these material domains similarly to the original publication on this subject [4], but including history-dependent behavior [14]. We consider four different materials with different porous microstructures but the same matrix phase – a simple von Mises plasticity model. This canonical example can be extended to more complex cases, including realistic microstructures, as shown in case study 2.

We present two case studies to demonstrate the usefulness of cooperative data-driven modeling (CDDM). The aim of CDDM is to create a model that is capable of performing multiple tasks but that also uses past knowledge from different tasks to decrease the number of training points required to learn a new task. Therefore, CDDM should achieve better performance than conventional training (without cooperation) for the same number of training points. In addition, the fact that CDDM uses the same neural network should make it more efficient in terms of the number of new parameters needed to learn a new task.

### 3.1. Data generation

An elastoplastic von Mises material with hardening is investigated to create a path-dependent problem. A fixed-sized square is utilized as a domain and holes of varying sizes and locations are placed inside the domain to create different tasks (see Fig. 2). The tasks originating from different domains can be seen in Fig. 3. For all the tasks the bottom part of the domain is fixed and the top part is deformed according to a uniform displacement in $u_{x_{1:t}}$ and $u_{y_{1:t}}$. We reinforce that Case Study 1 considers simpler finite element analysis because we want to facilitate the dissemination of the methodology and use fully open-source software — facilitating replication of the example and easy adaptation to other scenarios.

After applying the boundary conditions, the average Cauchy strain ($\bar{\boldsymbol{\varepsilon}}_{1:t}$) and Cauchy stress ($\bar{\boldsymbol{\sigma}}_{1:t}$) measures are obtained for the deformation path. Then the learning problem for a single task can be defined as,

$$\bar{\boldsymbol{\sigma}}_{1:t} = f(\bar{\boldsymbol{\varepsilon}}_{1:t}), \tag{5}$$

where a machine learning model is utilized to find the relationship $f : \bar{\boldsymbol{\varepsilon}}_{1:t} \mapsto \bar{\boldsymbol{\sigma}}_{1:t}$, which is a supervised regression problem with the history of the average strain components as an input and the average stress components as an output.

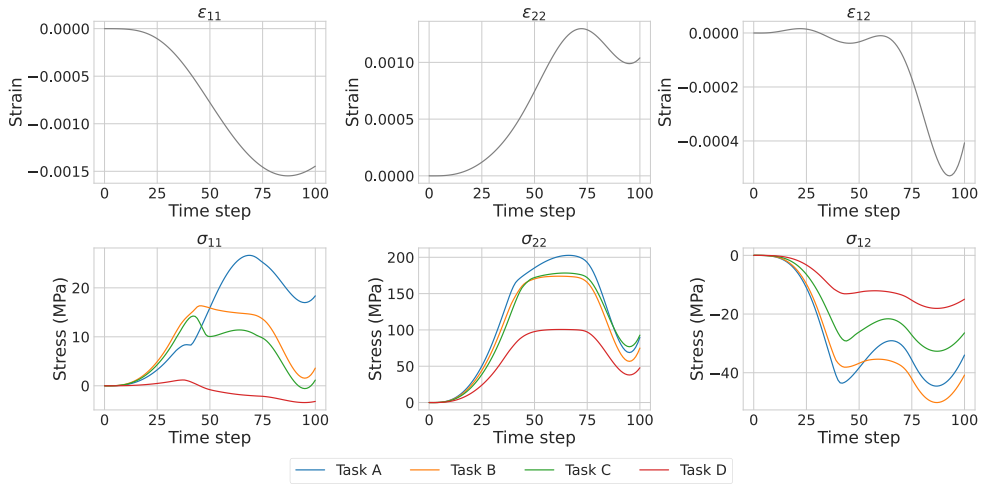



**Fig. 4.** An example of strain and stress paths.

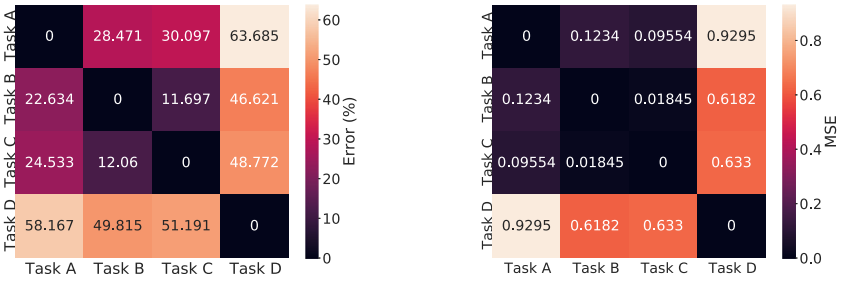



**Fig. 5.** The difference between the stress train data for considered tasks, calculated using the error measurement (left) and the MSE measurement (right).

The four material domains are subjected to the same 1000 paths of deformation. These 1000 paths are obtained from 100 end displacement values of the top boundary of the domain that are sampled from a Gaussian Process posterior that is conditioned on 20 displacement values sampled from a uniform distribution for each path. Then the average stress obtained for each path is calculated via the finite element method — see for example Fig. 4. The dataset is made available, and the simulations used to obtain it can be replicated via the source code. We highlight that different strategies for sampling the paths of plasticity material laws have been proposed [71] and that this choice can affect the number of paths needed to train the neural network up to the desired accuracy. Nevertheless, each presented task was subjected to the same deformation paths to calculate the domain-specific average stress and strain values. The data is generated using FEniCS [72].

In order to illustrate the difference in stresses obtained for the four tasks, Fig. 5 shows the error according to Eq. (6) and the mean-squared-error (MSE) between stresses for the training data for all tasks. We scale all the data since the model receives it in a scaled format. Here we use standard scaling where we remove the mean from every input feature and divide it by the standard deviation of the training set. The figure clarifies that Tasks A, B and C are more similar with each other than Task D.

### 3.2. Experiments setup

To evaluate the performance of CDDM we measure the error $E_i$ on every test path as follows:

$$E_i = \frac{1}{3}\left( \frac{||\boldsymbol{\sigma}^i_{11} - \hat{\boldsymbol{\sigma}}^i_{11}||_2}{||\boldsymbol{\sigma}^i_{11}||_2} + \frac{||\boldsymbol{\sigma}^i_{22} - \hat{\boldsymbol{\sigma}}^i_{22}||_2}{|\boldsymbol{\sigma}^i_{22}||_2} + \frac{||\boldsymbol{\sigma}^i_{12} - \hat{\boldsymbol{\sigma}}^i_{12}||_2}{||\boldsymbol{\sigma}^i_{12}||_2} \right) \cdot 100\%, \; i = 1, 2, \dots, N, \tag{6}$$

where $N$ is the number of testing points, $|\cdot|_2$ is the L2-norm, $\hat{\boldsymbol{\sigma}}^i_{11}, \hat{\boldsymbol{\sigma}}^i_{22}, \hat{\boldsymbol{\sigma}}^i_{12} \in \mathbb{R}^t$ are predicted stress components, and $\boldsymbol{\sigma}^i_{11}, \boldsymbol{\sigma}^i_{22}, \boldsymbol{\sigma}^i_{12} \in \mathbb{R}^t$ are the test ones.

Then, we compute the average over all $N$ test points to compute the final test error:

$$Err = \frac{1}{N} \sum_{i=1}^{N} E_i. \tag{7}$$

**Table 1**
Training hyperparameters.

| Training epochs | Learning rate | Weight decay | Pruning iterations | Pruning parameter $\alpha$ | Retraining epochs |
|---|---|---|---|---|---|
| 1000 | 0.01 | $10^{-6}$ | 1 | 0.95 | 200 |

**Table 2**
Test error (%) averaged over four considered task orders.

| | Task 1 | Tasks 2–4 | | | |
|---|---|---|---|---|---|
| | 800 paths | 800 paths | 400 paths | 200 paths | 100 paths |
| Standard training | **2.02** | 2.02 | 2.82 | 3.75 | 6.43 |
| CDDM | 2.14 | **1.92** | **2.18** | **2.84** | **3.97** |

The hyperparameters that we use to train a neural network are shown in Table 1. We use Adam [73] optimizer to train the model with the mean-squared-error (MSE) loss function. To prevent overfitting, we add weight decay regularization [74,75] to the networks' parameters, for which the updating rule is defined as follows:

$$w \leftarrow (1-\lambda)w - \alpha \frac{\partial \mathcal{L}_{\text{MSE}}}{\partial w}, \tag{8}$$

where $\alpha$ is the learning rate, $\lambda$ is the weight decay parameter and $\mathcal{L}_{\text{MSE}}$ is the loss on the current data batch. However, PyTorch implementation of weight decay for Adam optimizer refers to $L_2$ regularization [75].

As it is common in continual learning literature [76], we test the approach with different task orderings. Overall, we consider four orderings for the case of four tasks in a sequence:

- ordering 1: Task A → Task B → Task C → Task D;
- ordering 2: Task B → Task D → Task A → Task C;
- ordering 3: Task C → Task A → Task D → Task B;
- ordering 4: Task D → Task C → Task B → Task A.

### 3.3. Results

Firstly, we train the GRU with 2 cells and a hidden size of 128 in the sequence of four tasks. We train the first task with 800 training paths, and for each of the following tasks, we consider the cases of 800, 400, 200 and 100 training paths. We compare these results with the conventional case (non-cooperative) where every new task is trained with the same GRU but independently of the other tasks. Fig. 6 shows this comparison, where the blue bars refer to the cooperative model (CDDM) and the orange ones to the conventional case (standard or non-cooperative training). The test error is computed using Eq. (6). It is clear that CDDM significantly outperforms standard training when we decrease the number of training points. This effect is consistent across all four orders, independently of which task is considered to be the first. The main advantage of CDDM is that the pretrained parameters have an accumulative effect on future tasks. This multi-transfer effect has more significance under the low-data regime (e.g., 100 training paths). Also, the set of parameters depends on the order in which the tasks are learned.. In Table 2, we present the average error for every task when considering a different number of training points (number of paths); note that this error is the average over the four task orderings. We can clearly see that CDDM performs better than standard training for tasks 2–4 on average.

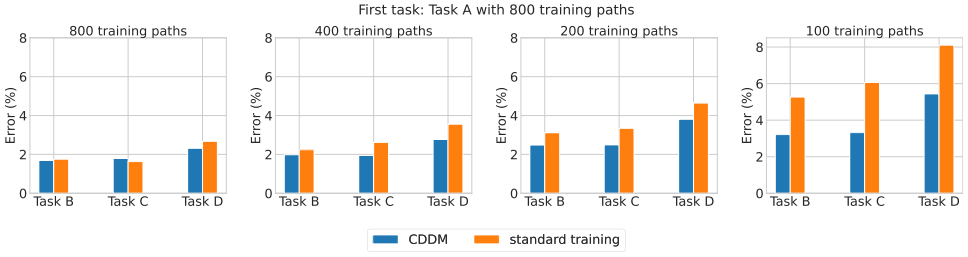
Moreover, we learn all four tasks with one network, while four separate networks are necessary for the conventional case. Therefore, using fewer parameters and achieving better performance. As clarified next, this is explained by the knowledge transfer that happens between subnetworks. Also, it should be noted that the network still has free space to learn future tasks, although saturation would occur soon if more tasks were considered because the neural network is small.

In Fig. 7, we show the prediction of CDDM with 200 training paths for tasks 2–4 respectively; the first task is learned with 800 paths. We compare CDDM with standard (non-cooperative) training with one network per task. As the figure illustrates, CDDM learns the data better than conventional training and requires just one network instead of four. Hence, Fig. 7 justifies our hypothesis of using continual learning as a possible solution to tackle the data scarcity problem in history-dependent constitutive law modeling. From this figure, it is clear that the GRU is able to learn material behavior by having 200 training paths for tasks 2–4.

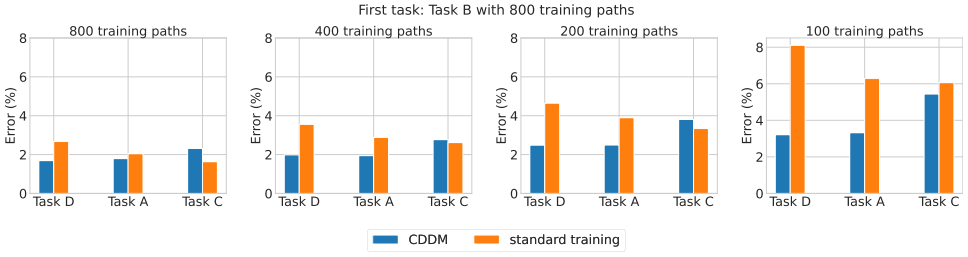
Overall, we observe that the continual learning strategy allows us not only to learn four different geometries with one network but also improves test error under the limited data regime. In addition, we can see that even if we have enough data, continual learning does not worsen the results significantly compared to standard training (no more than 0.5% difference).

## 4. Second case study: RVEs with periodic boundary conditions

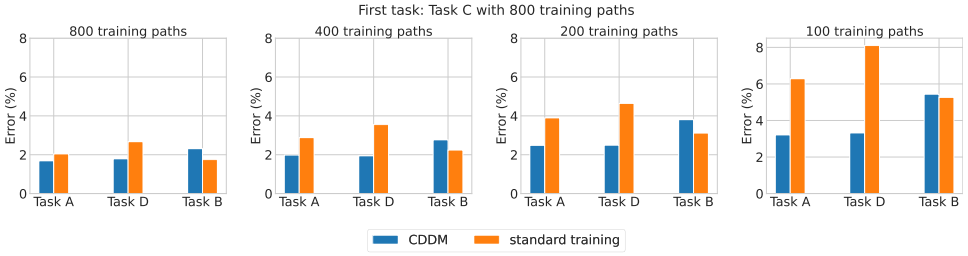
In this second case study, we apply the CDDM strategy to more realistic representative volume elements (RVEs) subjected to periodic boundary conditions, as thoroughly discussed in a past work [4]. The data is generated with a commercial finite element



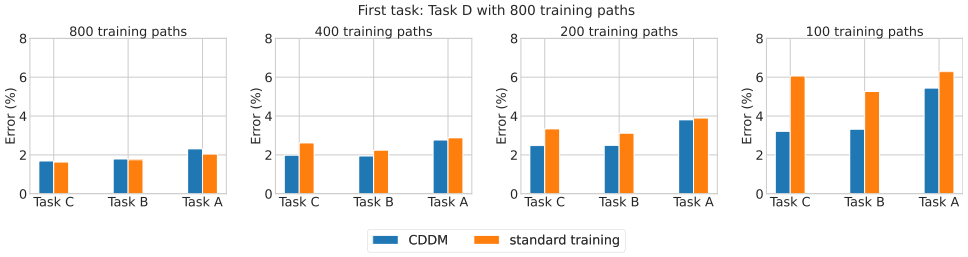

(a) Ordering 1.




(b) Ordering 2.




(c) Ordering 3.




(d) Ordering 4.

**Fig. 6.** First case study: CDDM results on orderings 1–4. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

solver, but both the code to generate the data as well as the dataset itself are made available. The RVE average stress–strain response is influenced by the microstructure, properties of each material phase, and loading conditions (average strain path that is converted into a periodic displacement at the boundary). Four different RVEs are considered in order to create four tasks. Each task pertains to learning the homogenized plasticity constitutive behavior (average stress–strain response) of a corresponding RVE. These RVEs were created such that they share some similarities and significant differences by considering different microstructures and material properties while applying the same average deformation paths to generate the dataset.

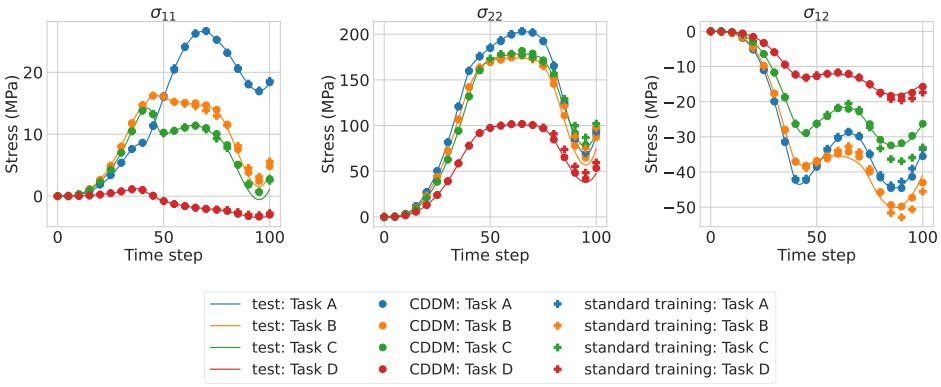



**Fig. 7.** First case study: 800 training paths for task 1 and 200 training paths for tasks 2–4.

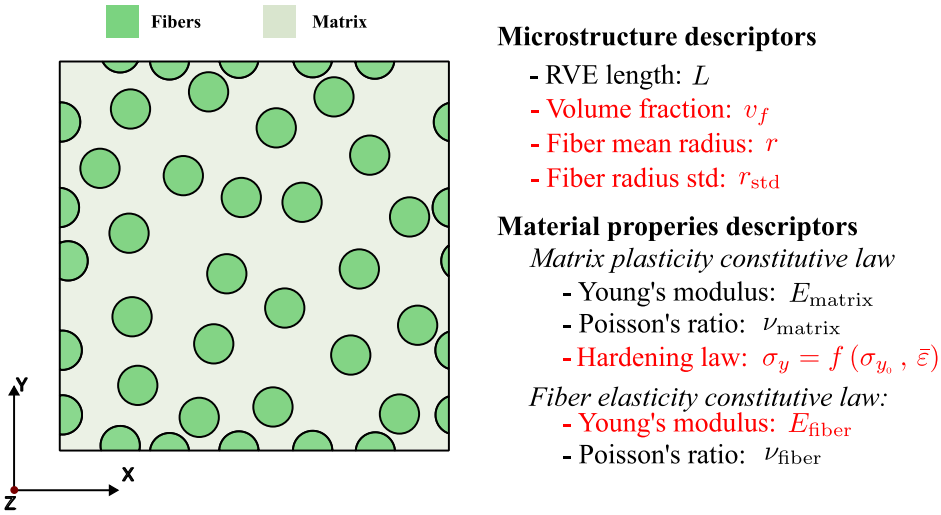



**Fig. 8.** Illustration of commonalities and discrepancies on setting up tasks.

### 4.1. RVEs simulation

Fig. 8 illustrates the type of two-dimensional composite material used to create the 4 RVEs (tasks). The two-phase RVEs are defined by 4 geometric descriptors: (1) RVE size, (2) fiber[3] volume fraction ($v_f$), (3) fiber mean radius ($r$), and (4) fiber radius standard deviation ($r_{\text{std}}$). The last two descriptors are used to create circular fibers whose radius is drawn from a Gaussian distribution with the corresponding mean and standard deviation. The descriptors of the material properties are simply the elastic properties of the fibers and matrix (Young's modulus ($E$) and Poisson's ratio ($\nu$)), and the plasticity properties of the matrix (isotropic hardening law that depends on the yield stress that completely defines the von Mises yield surface). These descriptors are all defined on the right part of Fig. 8 where the red font indicates the descriptors that are changed among the 4 RVEs and where the ones in black font indicate parameters that are fixed in this investigation, without loss of generality. The 4 RVEs (i.e. Tasks) are labeled A, B, C and D and the corresponding values for the descriptors are included in Table 3. For clarity, the 3 microstructures defining the RVEs (note that two RVEs have the same microstructure) are shown in Fig. 9.

As in Case Study 1, the target is to learn the average Cauchy stress ($\bar{\boldsymbol{\sigma}}_{1:t}$) which is dependent on the applied average strain path ($\bar{\boldsymbol{\varepsilon}}_{1:t}$). Therefore, the learning problem can be defined as: $\bar{\boldsymbol{\sigma}}_{1:t} = f(\bar{\boldsymbol{\varepsilon}}_{1:t})$, where $t$ is the pseudo-time step (load step) in the simulation defined to be 100. Meanwhile, 1000 different strain paths are generated according to a simple interpolation method. Specifically, for each average strain component ($\bar{\varepsilon}_{11}$, $\bar{\varepsilon}_{22}$, and $\bar{\varepsilon}_{12}$) of a path, 8 equally spaced points are sampled within the strain path, and the quadratic interpolation method is adopted to generate the full strain path. Then, the strain path is converted into a boundary value problem of the RVE and the finite element prediction is conducted with the commercial software ABAQUS [77] to simulate the corresponding average stress for different tasks. In Fig. 10, we present the difference between data computed with error metric (Eq. (6)) and the mean squared error (MSE).

[3] We consider plane strain conditions, so we tend to view the reinforcement phase as fibers instead of particles.

**Table 3**
Parameters configuration of different tasks (Units:SI(mm)).

| Task | Microstructure parameters | | | Hardening law | $E_{\text{fiber}}$ | Fixed parameters | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $v_f$ | $r$ | $r_{\text{std}}$ | | | size | $E_{\text{matrix}}$ | $\nu_{\text{matrix}}$ | $\nu_{\text{fiber}}$ |
| A | 0.45 | 0.01 | 0.003 | $\sigma_y = 0.5 + 0.5\bar{\varepsilon}$ | 10 | 0.048 | 100 | 0.30 | 0.19 |
| B | 0.30 | 0.003 | 0.0 | $\sigma_y = 0.5 + 0.5(\bar{\varepsilon})^{0.4}$ | 1 | | | | |
| C | 0.15 | 0.0015 | 0.0003 | $\sigma_y = 0.5(1 + \bar{\varepsilon})^{\frac{1}{0.4}}$ | 1000 | | | | |
| D | 0.30 | 0.003 | 0.0 | $\sigma_y = 3.0 + 0.5(\bar{\varepsilon})^{0.4}$ | 1 | | | | |

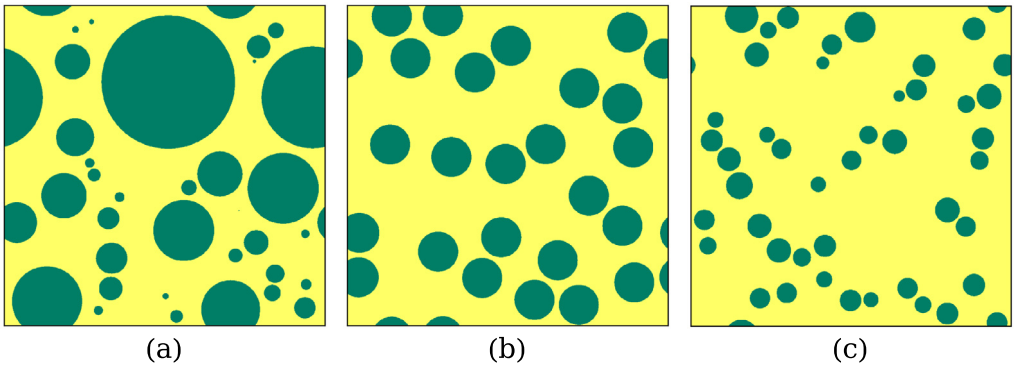



**Fig. 9.** Schematics of different microstructure configurations: (a) $v_f = 0.45$, $r = 0.01$, $r_{\text{std}} = 0.003$; (2) $v_f = 0.30$, $r = 0.003$, $r_{\text{std}} = 0.0$; (c) $v_f = 0.15$, $r = 0.0015$, $r_{\text{std}} = 0.0003$.

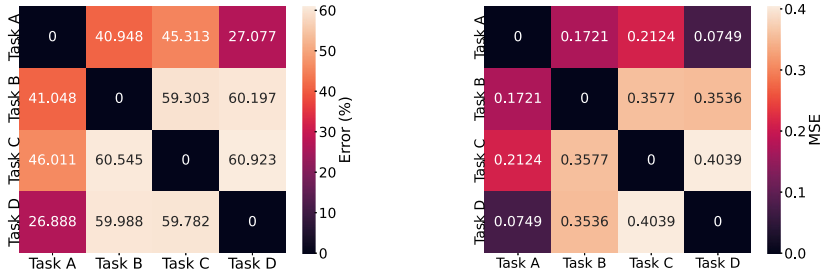



**Fig. 10.** The difference between the stress train data for considered RVE tasks, calculated using the error measurement (left) and the MSE measurement (right).

### 4.2. Results

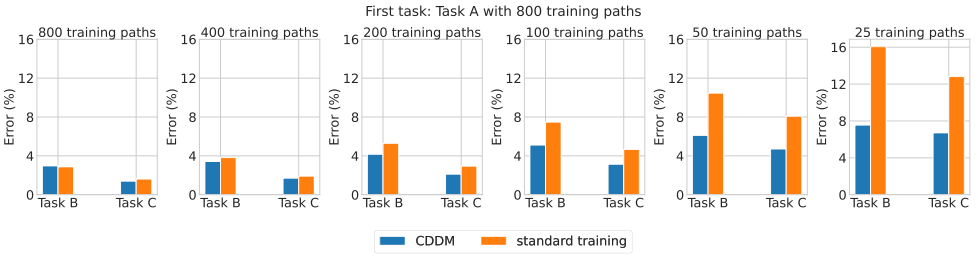
First, we train GRU on tasks A, B and C with the same hyperparameters as in Section 3. We consider three different tasks orders:

- ordering 1: Task A → Task B → Task C;
- ordering 2: Task C → Task A → Task B;
- ordering 3: Task B → Task C → Task A.

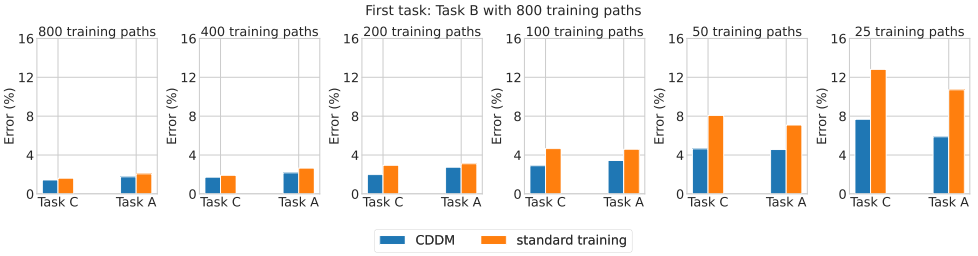
In Fig. 11, we show the results for these three task orderings. It is clear that with the decrease in the number of training paths, CDDM starts to outperform conventional training.

In Fig. 12, we present the prediction of one stress path with CDDM and standard training. The first task (task B) is trained with 800 training paths, while the next two tasks (tasks C and A) are trained using 200 training paths (**left**) and 25 training paths (**right**). If tasks C and A are learned with 200 paths, both CDDM and standard training predict stress well, however, CDDM does this with a single network. We observe that if GRU learns tasks in the cooperative approach, the prediction is more accurate than with conventional training if 25 training paths are given for the second and third tasks.

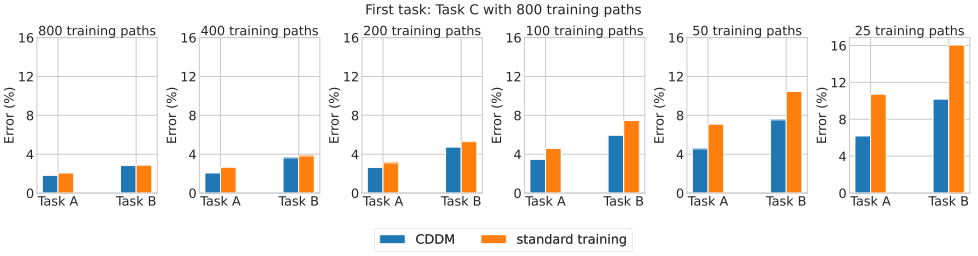
However, in an attempt to explore and report on the limitations of the presented method, we also investigated what occurs when we add task D (see Fig. 13) where an RVE has much larger yield stress (see Table 3), i.e. where the yield stress of the matrix becomes 3.0 MPa instead of 0.5 MPa as in the other tasks. In this case, the plastic response of the RVE is delayed, and we noticed that when Task D is learned first then there would be no advantage in learning cooperatively — see Fig. 10b. However, if the ordering is different, as shown in Fig. 10a, then the proposed cooperative model is still better. We think it is important to be clear that there might be situations in which the ordering of tasks actually leads to difficulties in learning cooperatively.




(a) Ordering 1.




(b) Ordering 2.




(c) Ordering 3.

**Fig. 11.** Second case study: CDDM results on orderings 1–3.

**Table 4**
Architectures comparison.

| Architecture | (1, 256) | (2, 128) | (3, 64) |
|---|---|---|---|
| The number of parameters | 201K | 151K | 63K |

## 5. Discussion

We want to highlight that all the above-mentioned results for both case studies are robust to the hyperparameter choice. For simplicity of presenting the previous results, they refer to a particular architecture configuration. However, in this section we elaborate on the robustness of the CDDM approach presented herein by considering different GRU architectures, varying the number of units and hidden state size. In addition, we elaborate on the knowledge transfer effect and analyze the portion of parameters shared between tasks and the ones dedicated only to one task.

### 5.1. Architectures comparison

In this section, we explore how CDDM depends on different GRU architecture hyperparameters such as the number of units or hidden states size. To begin, we consider the first learning case study (Section 3) with the corresponding four tasks: the first task is trained with 800 training paths and for the second task we vary the number of training points from 800 to 50. For GRU, we change the number of units from 1 to 3 and consider the hidden state size equal to 64, 128, and 256. Corresponding numbers of learnable parameters are shown in Table 4. In Fig. 14, we compare these three network configurations. Overall, we observe similar performance for all of these architectures with insignificant differences. From the figure, we observe that all architectures give us similar results, therefore CDDM is not limited to some special network configuration.

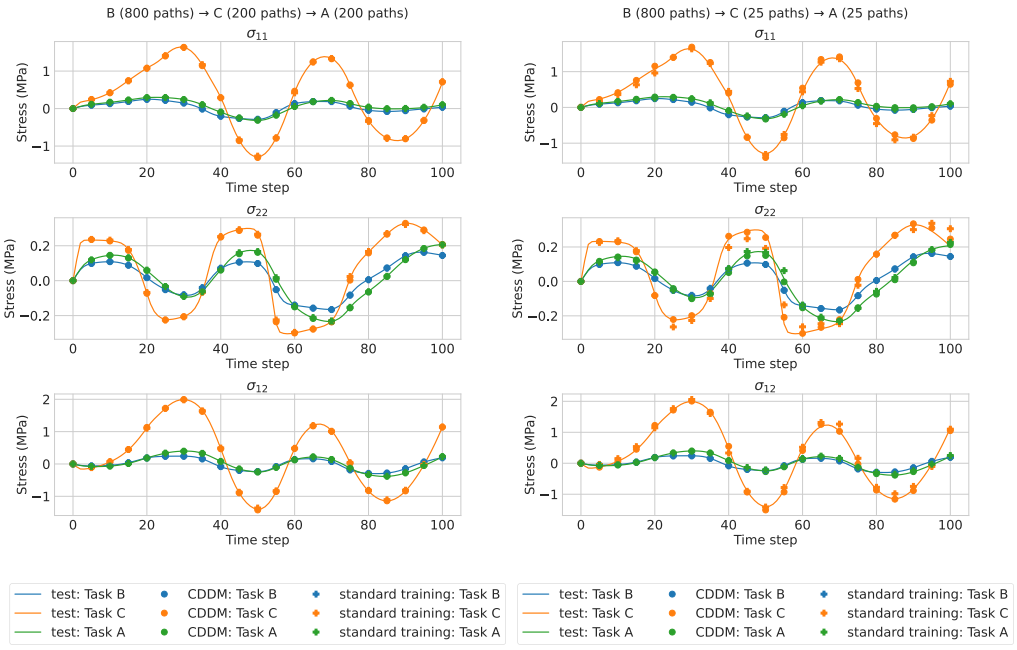
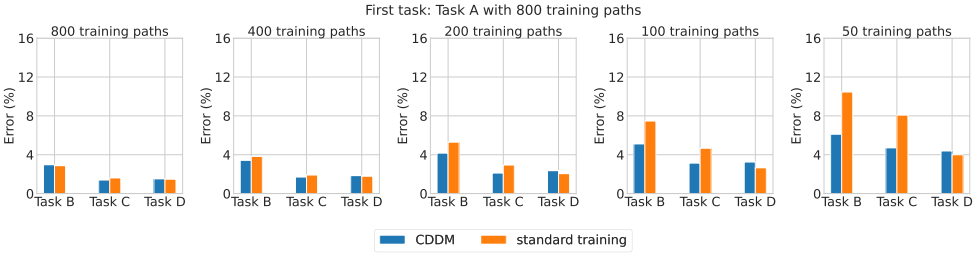
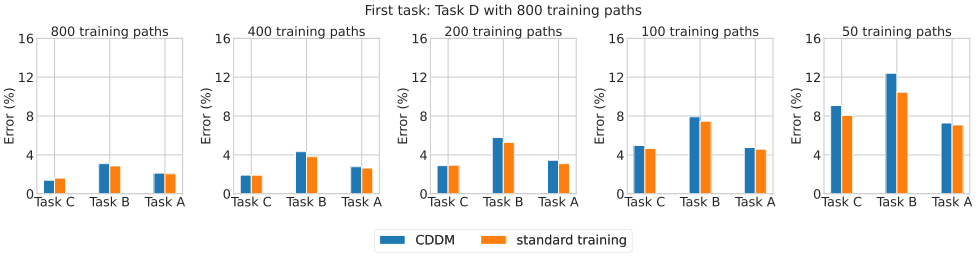



**Fig. 12.** Second case study: Comparison of the CDDM and standard training predictions with different numbers of training paths.




(a) Task A → Task B → Task C → Task D.




(b) Task D → Task C → Task B → Task A.

**Fig. 13.** Second case study: CDDM results on the sequences of four tasks.

We also want to note that smaller networks (fewer parameters) do not predict the material behavior better when considering the conventional training scenario (non-cooperative). To illustrate this, consider the GRU with 1 cell and the hidden state size of 64 which results in 13K parameters. We train this network on the second case study data and consider average error on tasks A – D using 800, 400, 200, 100, 50 and 25 training paths. The test errors for these cases are 2.09%, 2.74%, 3.59%, 5.2%, 7.73%, 12.9%. On the other hand, we consider the model with 2 cells and the hidden size of 128 (151K parameters, see Table 4), for which the test errors are 2.01%, 2.54%, 3.35%, 4.84%, 7.40%, 11.34%. As can be seen, the larger model has lower test error.

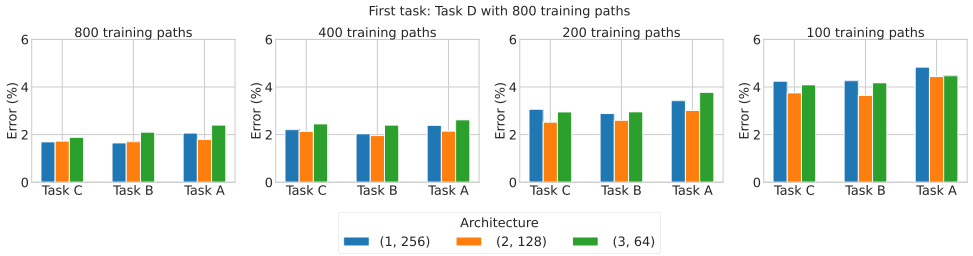
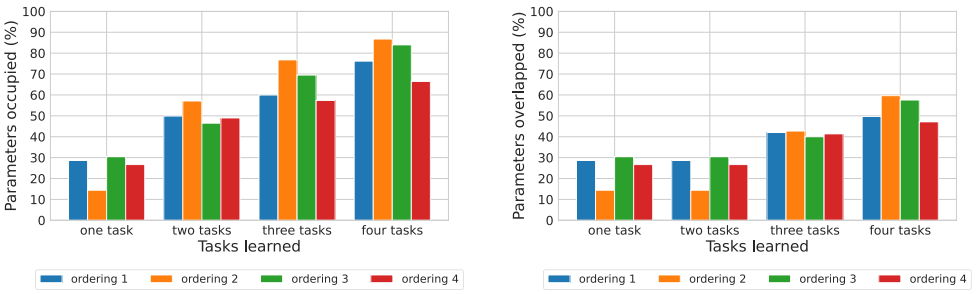



**Fig. 14.** First case study: architectures comparison on ordering 4.




(a) Occupied parameters. (b) Shared parameters.

**Fig. 15.** First case study: subnetworks analysis: occupied and shared parameters.

### 5.2. Knowledge transfer

A crucial characteristic necessary to establishing the CDDM paradigm is the robustness to different task orderings, both in terms of prediction error and the number of parameters used for every task. Focusing on the first case study, we show the relation between subnetworks in the case where the first task is trained with 800 paths and all the other tasks with 200 training paths. In Fig. 15a, we show which percentage of the total number of parameters is occupied after a new task is learned. We compare these percentages across four orderings, and in general, we observe the consistency in the number of used parameters with insignificant differences. This means that the ordering of tasks has a negligible effect on how many parameters are occupied in the end.

At the same time, in Fig. 15b, we demonstrate the percentage of shared connections while the model learns a new task. So, for instance, we observe in ordering 3 that up to 60% of parameters are assigned to more than one subnetwork when all the tasks are learned. Overall, at least 45% of the parameters are shared between multiple tasks without a negative impact on model performance. From the figure, it is clear that the changes in the numbers of shared parameters are consistent across all orderings, illustrating robustness to different task sequences. Nevertheless, after experimenting with many tasks and considering two different case studies, we were able to find one task ordering for Case Study 2 where the cooperative data-driven modeling process was not beneficial when compared to learning all tasks separately (recall Fig. 10b).

## 6. Conclusion and future directions

This work introduces the concept of continual learning and the notion of cooperative data-driven modeling. We focus on solid mechanics applications by considering two case studies involving history-dependent plasticity problems. To the best of our knowledge, this is the first example of the application of continual learning in Mechanics and among the first in Engineering applications. We demonstrate that a recurrent neural network can sequentially learn multiple tasks, without replaying data from previous tasks and without forgetting — an important distinction when comparing to transfer learning methods, and a key enabler of cooperative modeling. The proposed method is based on creating task-related subnetworks that transfer knowledge from each other by sharing neural connections. This is demonstrated to decrease the number of training data required to learn a new task (in this case, a new material law). The approach is robust to different task orders.

As a final note, the authors share their belief that the proposed cooperative data-driven modeling concept has a lot of potential for future developments. More efficient ways of sharing knowledge or selecting subnetworks (when needed), delaying the premature saturation of the network, and accelerating training are only a few possibilities to improve the proposed strategy. Notwithstanding, the prospect of fostering collaborations across different research communities by taking advantage of a machine learning model from a group and adding new capabilities to it such that it solves a new task using less training data and without forgetting how to perform the original task is an exciting new development.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

The data and code relative to this research are available as open-access in a public repository. A footnote was added to the article with the link.

## Acknowledgments

Miguel A. Bessa acknowledges the support from the NWO, The Netherlands Veni award 'Artificial intelligence towards a sustainable future: ecodesign of recycled polymers and composites' (with project number 17260 of the research programme Applied and Engineering Sciences) which is financed by the Dutch Research Council (NWO), The Netherlands. Jiaxiang YI acknowledges the generous support from the China Scholarship Council (CSC).

## Appendix. Summary of the pruning algorithm (NNrelief) used by the CP&S method

NNrelief prunes connections with the smallest contribution in the signal of the following neuron. For the incoming signal $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ with $N$ data points $\mathbf{x}_n = (x_{n1}, \dots, x_{nm_1}) \in \mathbb{R}^{m_1}$, we compute the importance scores:

$$s_{ij}(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N) = \frac{\overline{|w_{ij}x_i|}}{\sum_{k=1}^{m_1} \overline{|w_{kj}x_k|} + |b_j|}, \tag{A.1}$$

where $\overline{|w_{ij}x_i|} = \frac{1}{N}\sum_{n=1}^{N}|w_{ij}x_{ni}|$ and $\mathbf{W} = (w_{ij}) \in \mathbb{R}^{m_1 \times m_2}$ is a corresponding weight matrix, $\mathbf{b} = (b_1, b_2, \dots, b_{m_2})^T \in \mathbb{R}^{m_2}$ is a bias vector. Importance score for the bias of the neuron $j$ is computed by $s_{m_1+1,j} = \frac{|b_j|}{\sum_{k=1}^{m_1}|w_{kj}x_k|+|b_j|}$. The connections with the smallest importance are pruned. Algorithm 1 summarizes the procedure for feedforward layers.

**Algorithm 1** Pseudocode for NNrelief

**Require:** network $\mathcal{N}$, training dataset $\mathbf{X}$, pruning hyperparameter $\alpha$.

$\mathbf{X}^{(0)} \leftarrow \mathbf{X}$
**for** every layer $l = 1, \dots, L$ **do**
  $\mathbf{X}^{(l)} \leftarrow \text{layer}\left(\mathbf{X}^{(l-1)}\right)$
  **for** every neuron $j$ in layer $l$ **do**
    Compute importance scores $s_{ij}^{(l)}$ for every incoming connection $w_{ij}$ and bias $b_j$ using Eq. (A.1).
    $\hat{s}_{ij}^{(l)} \leftarrow \text{Sort}(s_{ij}^{(l)}, order = descending)$.
    Find $p_0 = \min\{p : \sum_{i=1}^{p} \hat{s}_{ij}^{(l)} \geq \alpha\}$.
    Prune connections with importance score $s_{ij}^{(l)} < \hat{s}_{p_0 j}^{(l)}$.
  **end for**
**end for**

## References

[1] S. Shanmuganathan, Artificial neural network modelling: An introduction, in: Artificial Neural Network Modelling, Springer, 2016, pp. 1–14.
[2] T. Wuest, D. Weimer, C. Irgens, K.-D. Thoben, Machine learning in manufacturing: advantages, challenges, and applications, Prod. Manuf. Res. 4 (1) (2016) 23–45.
[3] J. Schmidt, M.R. Marques, S. Botti, M.A. Marques, Recent advances and applications of machine learning in solid-state materials science, npj Comput. Mater. 5 (1) (2019) 1–36.
[4] M. 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, http://dx.doi.org/10.1016/j.cma.2017.03.037.
[5] G. Capuano, J.J. Rimoli, Smart finite elements: A novel machine learning application, Comput. Methods Appl. Mech. Engrg. 345 (2019) 363–381.
[6] P. Thakolkaran, A. Joshi, Y. Zheng, M. Flaschel, L. De Lorenzis, S. Kumar, NN-EUCLID: deep-learning hyperelasticity without stress data, 2022, arXiv preprint arXiv:2205.06664.
[7] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (6) (2021) 422–440.
[8] A. Dekhovich, D.M. Tax, M.H. Sluiter, M.A. Bessa, Continual prune-and-select: Class-incremental learning with specialized subnetworks, 2022, arXiv preprint arXiv:2208.04952.
[9] J. Ghaboussi, J.H. Garrett, X. Wu, Knowledge-based modeling of material behavior with neural networks, J. Eng. Mech. 117 (1) (1991) 132–153, http://dx.doi.org/10.1061/(ASCE)0733-9399(1991)117:1(132).
[10] R. Ibanez, E. Abisset-Chavanne, J.V. Aguado, D. Gonzalez, E. Cueto, F. Chinesta, A manifold learning approach to data-driven computational elasticity and inelasticity, Arch. Comput. Methods Eng. 25 (2018) 47–57.

[11] R. Jones, J. Templeton, C. Sanders, J. Ostien, Machine learning models of plastic flow based on representation theory, CMES-Comput. Model. Eng. Sci. 117 (3) (2018).
[12] L.T.K. Nguyen, M.-A. Keip, A data-driven approach to nonlinear elasticity, Comput. Struct. 194 (2018) 97–115.
[13] I. Rocha, P. Kerfriden, F. van der Meer, On-the-fly construction of surrogate constitutive models for concurrent multiscale mechanical analysis through probabilistic machine learning, J. Comput. Phys.: X 9 (2021) 100083, http://dx.doi.org/10.1016/j.jcpx.2020.100083.
[14] M. Mozaffar, R. Bostanabad, W. Chen, K. Ehmann, J. Cao, M.A. Bessa, Deep learning predicts path-dependent plasticity, Proc. Natl. Acad. Sci. 116 (52) (2019) 26414–26420, http://dx.doi.org/10.1073/pnas.1911815116.
[15] F. Ghavamian, A. Simone, Accelerating multiscale finite element simulations of history-dependent materials using a recurrent neural network, Comput. Methods Appl. Mech. Engrg. 357 (2019) 112594.
[16] N.N. Vlassis, R. Ma, W. Sun, Geometric deep learning for computational mechanics part i: Anisotropic hyperelasticity, Comput. Methods Appl. Mech. Engrg. 371 (2020) 113299.
[17] D.W. Abueidda, S. Koric, N.A. Sobh, H. Sehitoglu, Deep learning for plasticity and thermo-viscoplasticity, Int. J. Plast. 136 (2021) 102852.
[18] J.N. Fuhg, N. Bouklas, The mixed deep energy method for resolving concentration features in finite strain hyperelasticity, J. Comput. Phys. 451 (2022) 110839.
[19] B. Liu, N. Kovachki, Z. Li, K. Azizzadenesheli, A. Anandkumar, A.M. Stuart, K. Bhattacharya, A learning-based multiscale method and its application to inelastic impact problems, J. Mech. Phys. Solids 158 (2022) 104668.
[20] F. Masi, I. Stefanou, P. Vannucci, V. Maffi-Berthier, Thermodynamics-based artificial neural networks for constitutive modeling, J. Mech. Phys. Solids 147 (2021) 104277.
[21] F. As' ad, P. Avery, C. Farhat, A mechanics-informed artificial neural network approach in data-driven constitutive modeling, Internat. J. Numer. Methods Engrg. 123 (12) (2022) 2738–2759.
[22] A. Zhang, D. Mohr, Using neural networks to represent von Mises plasticity with isotropic hardening, Int. J. Plast. 132 (2020) 102732.
[23] P. Saidi, H. Pirgazi, M. Sanjari, S. Tamimi, M. Mohammadi, L.K. Béland, M.R. Daymond, I. Tamblyn, Deep learning and crystal plasticity: A preconditioning approach for accurate orientation evolution prediction, Comput. Methods Appl. Mech. Engrg. 389 (2022) 114392.
[24] C. Bonatti, B. Berisha, D. Mohr, From CP-FFT to CP-RNN: Recurrent neural network surrogate model of crystal plasticity, Int. J. Plast. (2022) 103430.
[25] Z. Liu, C. Wu, M. Koishi, A deep material network for multiscale topology learning and accelerated nonlinear modeling of heterogeneous materials, Comput. Methods Appl. Mech. Engrg. 345 (2019) 1138–1168.
[26] G.C. Peng, M. Alber, A. Buganza Tepole, W.R. Cannon, S. De, S. Dura-Bernal, K. Garikipati, G. Karniadakis, W.W. Lytton, P. Perdikaris, et al., Multiscale modeling meets machine learning: What can we learn? Arch. Comput. Methods Eng. 28 (3) (2021) 1017–1037.
[27] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, Adv. Neural Inf. Process. Syst. 30 (2017).
[28] P. Dütting, Z. Feng, H. Narasimhan, D. Parkes, S.S. Ravindranath, Optimal auctions through deep learning, in: International Conference on Machine Learning, PMLR, 2019, pp. 1706–1715.
[29] M.D. Wilkinson, M. Dumontier, I.J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L.B. da Silva Santos, P.E. Bourne, et al., The FAIR Guiding Principles for scientific data management and stewardship, Sci. Data 3 (1) (2016) 1–9.
[30] C. Draxl, M. Scheffler, NOMAD: The FAIR concept for big data-driven materials science, Mrs Bull. 43 (9) (2018) 676–682.
[31] A. Jacobsen, R. de Miranda Azevedo, N. Juty, D. Batista, S. Coles, R. Cornet, M. Courtot, M. Crosas, M. Dumontier, C.T. Evelo, et al., FAIR principles: interpretations and implementation considerations, Data Intell. 2 (1–2) (2020) 10–29.
[32] M. McCloskey, N.J. Cohen, Catastrophic interference in connectionist networks: The sequential learning problem, in: Psychology of Learning and Motivation, Vol. 24, Elsevier, 1989, pp. 109–165.
[33] R.M. French, Catastrophic forgetting in connectionist networks, Trends Cogn. Sci. 3 (4) (1999) 128–135.
[34] I.J. Goodfellow, M. Mirza, D. Xiao, A. Courville, Y. Bengio, An empirical investigation of catastrophic forgetting in gradient-based neural networks, 2013, arXiv preprint arXiv:1312.6211.
[35] R. Caruana, Learning many related tasks at the same time with backpropagation, Adv. Neural Inf. Process. Syst. 7 (1994).
[36] R. Vilalta, Y. Drissi, A perspective view and survey of meta-learning, Artif. Intell. Rev. 18 (2) (2002) 77–95.
[37] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359, http://dx.doi.org/10.1109/TKDE.2009.191.
[38] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, C. Liu, A survey on deep transfer learning, in: International Conference on Artificial Neural Networks, Springer, 2018, pp. 270–279.
[39] Z. Liu, C.T. Wu, M. Koishi, Transfer learning of deep material network for seamless structure–property predictions, Comput. Mech. 64 (2) (2019) 451–465, http://dx.doi.org/10.1007/s00466-019-01704-4.
[40] E. Lejeune, B. Zhao, Exploring the potential of transfer learning for metamodels of heterogeneous material deformation, 2020, arXiv:2010.16260.
[41] S. Thrun, L. Pratt (Eds.), Learning to Learn, Springer US, Boston, MA, 1998, http://dx.doi.org/10.1007/978-1-4615-5529-2.
[42] F. Zenke, B. Poole, S. Ganguli, Continual learning through synaptic intelligence, in: International Conference on Machine Learning, PMLR, 2017, pp. 3987–3995.
[43] G.I. Parisi, R. Kemker, J.L. Part, C. Kanan, S. Wermter, Continual lifelong learning with neural networks: A review, Neural Netw. 113 (2019) 54–71.
[44] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, T. Tuytelaars, Memory aware synapses: Learning what (not) to forget, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 139–154.
[45] M. Biesialska, K. Biesialska, M.R. Costa-Jussa, Continual lifelong learning in natural language processing: A survey, 2020, arXiv preprint arXiv:2012.09823.
[46] G. Sokar, D.C. Mocanu, M. Pechenizkiy, Spacenet: Make free space for continual learning, Neurocomputing 439 (2021) 1–11.
[47] M. Masana, X. Liu, B. Twardowski, M. Menta, A.D. Bagdanov, J. van de Weijer, Class-incremental learning: survey and performance evaluation on image classification, 2020, arXiv preprint arXiv:2010.15277.
[48] M. Wortsman, V. Ramanujan, R. Liu, A. Kembhavi, M. Rastegari, J. Yosinski, A. Farhadi, Supermasks in superposition, Adv. Neural Inf. Process. Syst. (2020).
[49] Z. Li, D. Hoiem, Learning without forgetting, IEEE Trans. Pattern Anal. Mach. Intell. 40 (12) (2017) 2935–2947.
[50] A. Chaudhry, P.K. Dokania, T. Ajanthan, P.H. Torr, Riemannian walk for incremental learning: Understanding forgetting and intransigence, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 532–547.
[51] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, C.H. Lampert, icarl: Incremental classifier and representation learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 2001–2010.
[52] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, Y. Fu, Large scale incremental learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 374–382.
[53] A. Douillard, M. Cord, C. Ollion, T. Robert, E. Valle, Podnet: Pooled outputs distillation for small-tasks incremental learning, in: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16, Springer, 2020, pp. 86–102.
[54] A. Mallya, S. Lazebnik, Packnet: Adding multiple tasks to a single network by iterative pruning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 7765–7773.
[55] A. Mallya, D. Davis, S. Lazebnik, Piggyback: Adapting a single network to multiple tasks by learning to mask weights, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 67–82.
[56] S. Golkar, M. Kagan, K. Cho, Continual learning via neural pruning, 2019, arXiv preprint arXiv:1903.04476.

[57] A. Dekhovich, D.M. Tax, M.H. Sluiter, M.A. Bessa, Neural network relief: a pruning algorithm based on neural activity, 2021, arXiv preprint arXiv:2109.10795.
[58] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, M. Rastegari, What's hidden in a randomly weighted neural network? in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 11893–11902.
[59] S. Han, J. Pool, J. Tran, W.J. Dally, Learning both weights and connections for efficient neural networks, 2015, arXiv preprint arXiv:1506.02626.
[60] H. Hu, R. Peng, Y.-W. Tai, C.-K. Tang, Network trimming: A data-driven neuron pruning approach towards efficient deep architectures, 2016, arXiv preprint arXiv:1607.03250.
[61] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning Internal Representations by Error Propagation, Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
[62] M.I. Jordan, Serial order: A parallel distributed processing approach, in: Advances in Psychology, Vol. 121, Elsevier, 1997, pp. 471–495.
[63] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, S. Khudanpur, Recurrent neural network based language model, in: Interspeech, 2010, pp. 1045–1048.
[64] H. Sak, A.W. Senior, F. Beaufays, Long short-term memory recurrent neural network architectures for large scale acoustic modeling, in: Fifteenth Annual Conference of the International Speech Communication Association, 2014, pp. 338–342, http://dx.doi.org/10.21437/Interspeech.2014-80.
[65] S. Hochreiter, Untersuchungen zu Dynamischen Neuronalen Netzen, Vol. 91 (Diploma), Technische Universität München, 1991, (1).
[66] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neural Netw. 5 (2) (1994) 157–166.
[67] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[68] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, 2014, arXiv preprint arXiv:1406.1078.
[69] G. Chen, Recurrent neural networks (RNNs) learn the constitutive law of viscoelasticity, Comput. Mech. (2021) 11.
[70] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, Adv. Neural Inf. Process. Syst. 32 (2019).
[71] L. Wu, N.G. Kilingar, L. Noels, et al., 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. Engrg. 369 (2020) 113234.
[72] A. Logg, K.-A. Mardal, G. Wells (Eds.), Automated Solution of Differential Equations by the Finite Element Method, in: Lecture Notes in Computational Science and Engineering, vol. 84, Springer, Berlin, Heidelberg, 2012, http://dx.doi.org/10.1007/978-3-642-23099-8.
[73] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[74] S. Hanson, L. Pratt, Comparing biases for minimal network construction with back-propagation, Adv. Neural Inf. Process. Syst. 1 (1988).
[75] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, 2017, arXiv preprint arXiv:1711.05101.
[76] M. Masana, B. Twardowski, J. Van de Weijer, On class orderings for incremental learning, 2020, arXiv preprint arXiv:2007.02145.
[77] M. Smith, ABAQUS/Standard User's Manual, Version 6.9, Dassault Systèmes Simulia Corp, United States, 2009.