Received 10 October 2023, accepted 20 November 2023, date of publication 23 November 2023,
date of current version 29 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3336292

RESEARCH ARTICLE

A Comparative Study of Sparsity Promoting Techniques in Neural Network for Modeling Non-Linear Dynamics

EMIL JOHANNESEN HAUGSTVEDT, ALBERTO MIÑO CALERO, ERLEND TORJE BERG LUNDBY, (Member, IEEE), ADIL RASHEED, AND JAN TOMMY GRAVDAHL, (Senior Member, IEEE)
Department of Engineering Cybernetics, Norwegian University of Science and Technology, 7491 Trondheim, Norway

Corresponding author: Adil Rasheed (adil.rasheed@.ntnu.no)

This work was supported in part by the Towards Autonomy in Process Industries (TAPI) Project under Grant 294544, and in part by the Explainable AI Systems for Gradual Industry Adoption (EXAIGON) under Grant 304843.

ABSTRACT Sparsity-promoting techniques show promising results in improving the generalization of neural networks. However, the literature contains limited information on how different sparsity techniques affect generalization when using neural networks to model non-linear dynamical systems. This study examines the use of sparsity-enhancing techniques to improve accuracy and reduce the divergence rate of neural networks used to simulate such systems. A range of sparsity methods, including hard and soft thresholding, pruning and regrowing, and L1-regularization, were applied to neural networks and evaluated in a complex nonlinear aluminum extraction process by electrolysis. The results showed that the most effective technique was L1 regularization, which enhanced the important connections in the network and improved the model performance. In contrast, many of the more advanced sparsity techniques resulted in significantly worse performance and higher divergence rates. Additionally, the application of Stochastic Weight Averaging during training increased performance and reduced the number of diverging simulations. These findings suggest that carefully selecting the right sparsity techniques and model structures can improve the performance of neural network-based simulations of dynamical systems.

INDEX TERMS Aluminum electrolysis, data-driven modeling, nonlinear dynamics, ordinary differential equations, sparse neural networks.

I. INTRODUCTION

Deep Learning and neural networks have had a significant impact in many scientific and technological domains as varied as genetic expressions [1], hyperspectral image analysis [2] or smart fish farming [3]. In the field of Computer Vision neural networks have proven to excel in a wide range of tasks beyond image classification [4], such as image segmentation [5], object detection [6] or image restoration [7]. Moreover, recent generative models DALL-E 2 [8], [9] and ChatGPT [10] have changed the game in terms of possibilities with more powerful networks that generate data almost impossible to discern as "artificial".

The associate editor coordinating the review of this manuscript and approving it for publication was Thomas Canhao Xu.

Despite the success of Deep Learning and its flexibility to be used in many applications, its advantages have not been broadly explored in engineering problems such as modeling non-linear dynamical systems, for which more traditional analytical techniques, often more time-consuming and expensive, difficult to apply, and presumably, less accurate in highly variable settings, need to be applied. Despite the topic of combining neural networks and dynamical systems being introduced as early as 1990 [11], their application in this field is still a largely unexplored area. The successful integration of neural networks into highly variable non-linear dynamical systems has the potential to speed up simulations, increase accuracy, model more complex systems, or discover unknown hidden physics, among other advantages, motivating a more thorough investigation.

To understand the advantages that using neural networks to simulate dynamical systems can bring, we must acknowledge that dynamical systems are governed by differential equations and that they can be a challenge to model in certain settings. Some dynamical systems can be represented with linear equations in simple environments, but most of them are characterized by complex non-linear differential equations. Highly non-linear differential equations, when they are perfectly known, require the development of numerical simulators which require considerable resources in terms of time, cost, and human expertise not only to build them but to execute them as well. Moreover, often assumptions need to be made that are unrealistic, and the highly varying environments make the simulators deviate from the real trajectories and need constant revisions and adjustments. Other times, the dynamics of the system are not completely known, which makes the development of numerical simulators even more difficult or even unfeasible.

Neural networks, for their part, despite their vast popularity nowadays, also have drawbacks that may motivate why they are not popular to simulate dynamical systems and need to be handled. The absence of interpretability, the difficulty in assessing their reliability, or their generalization problems (overfitting) to unseen data are among those common drawbacks [12], [13]. Among the techniques that can help to fix these issues, we focus our attention on sparsification techniques. The fundamental idea of promoting sparsity in neural networks is that it helps the network gain generalizability [14], [15] by learning the most important and general structures in the data. This helps to avoid overfitting scenarios [16] in which the training data are presumably memorized and very specific patterns are given too much weight, causing poor generalizability to new data. Most of the regularization techniques are focused on this goal, but sparsity has a feature that is absent in the rest: it reduces the connections between neurons, hence smaller numbers of parameters. Sparse neural networks exhibit a smaller size compared to dense fully connected without losing performance, exhibit better generalization capabilities, and require less computer power. Moreover, the decrease in interconnections between neurons has the potential to produce networks with some level of interpretability, although it falls out of the scope of this work.

In simulating dynamical systems, rather than only regarding accuracy, it can be more important to have stable and reliable models. Sparsity techniques can be fundamental here because they promote parameters by ruling out less important ones. This supports the assumption that promoting sparsity may lead to learning the most general behaviors embedded in the training data, leveraging the bias-variance trade-off, and enforcing to a certain degree models that are less prone to diverge from real trajectories in the face of unseen data. For all these reasons, we believe that the use of sparse neural networks to simulate dynamical systems deserves further evaluation.

Although not their primary application, the use of neural networks to simulate dynamical systems can be found in the literature. In [17], an algorithm is introduced to develop general dynamical systems by synthesizing recurrent neural networks capable of working on continuous time. Another work in which recurrent neural networks are used to model dynamical systems is [18], and it presents how these networks are capable of handling complex non-linear dynamical systems and forecasting a time horizon of predictions in aircraft motion. A recent extensive literature review on the field [12] shows many ways in which neural networks can be used to simulate dynamical systems.

Sparsity in neural networks has been a very popular field recently. It generally focuses on how to reduce the number of parameters in a neural network model without sacrificing performance to reduce the memory usage and inference cost of large neural networks. For example, recent advances have allowed large image classification models to achieve significant reductions in the number of parameters without sacrificing accuracy [19], [20] [21]. These techniques work by identifying and amplifying the important connections and weights in the network while pruning the less important ones. A comprehensive review of state-of-the-art sparsity techniques can be found in [22].

There can also be found research about how inducing sparsity can be used in the context of dynamical systems [23], but sparse neural networks are only briefly referenced, and only due to l1 normalization, which is often applied as a regularization technique but can also be considered a method to sparsify neural networks. The authors of [24] show how to perform Sparse Identification of Nonlinear Dynamics with an algorithm called SINDy that promotes sparsity through sparse regression over candidate non-linear terms. Based on SINDy's framework, [25] extends it and combines it with Model Predictive Control, proposing the SINDy-MPC framework. In [26], sparse polynomial regression is employed together with neural networks to present an operator inference framework for dynamical systems. These works are examples of the use of sparse neural networks in the field of dynamical systems.

Works that combine sparse neural networks and dynamical systems are of the type of [27] and [28] and do not delve into simulating dynamical systems with neural networks. In [27], the authors investigate the efficiency of constructing graph neural networks by exploiting "the idea of representing each input graph as a fixed point of a dynamical system". In their approach, sparsity is studied by limiting the number of connections of each hidden neuron. In [28] sparsity is examined in the context of recurrent neural networks, but with the goal of recovering sparse input signals from the output of the networks, and the dynamics being investigated are those of the recurrent networks.

It can be noticed that the use of sparse neural networks to simulate dynamical systems cannot be found in the works referenced here or those presented in the literature surveys mentioned. To the best of our knowledge, we have not found literature that delves into it, remaining an underdeveloped area.

To address this absence, this work presents an empirical study on how to address neural network limitations by incorporating sparsity to improve the stability and generalizability of neural networks and how that can lead to the successful modeling of a complex non-linear dynamical system. To do so, we use popular well-established sparsification techniques with different complexities, some novel and some that have been applied for some time already: soft thresholding for learnable sparsity [20], magnitude pruning [19], and dynamic sparse reparameterization [21], which will be described in the next section.

The research questions we intend to answer with this work are as follows:

- By inducing sparsity to a neural network during training, is it possible to train neural networks that learn only the important and general parts of the dynamics of a dynamical system?
- Will the application of generalization techniques during training result in more stable simulations with better accuracy?
- Will feasible combinations of sparsity and generalization techniques together lead to more stable, generalizable, and accurate simulations conducted by Neural Networks?

The structure of this work is as follows. Section I presents an introduction with relevant works and the goal of this work, along with the research questions that will be answered. The theoretical knowledge and information to understand the ideas, experiments, and results of this work are presented in Section II, which includes the applied techniques. Section III provides the information needed to understand and replicate the experiments and results, explaining how the data is generated, how the neural networks are designed to apply the different sparsity and regularization techniques, which hyperparameters are explored, and what is the general design of the experiments performed. The results are presented in Section IV along with a discussion of the findings. Finally, Section V summarizes the main findings of the experiments and discusses potential directions for future research based on the results achieved.

II. THEORY

A. NON-LINEAR DYNAMICAL SYSTEM

Dynamical systems are continuous and discrete systems governed by differential equations [29]. Differential equations describe how systems evolve over time and are therefore important in many scientific and engineering applications. The two main categories of differential equations are ordinary differential equations (ODEs) and partial differential equations (PDEs) [29]. ODEs involve only a single independent variable, typically time, while PDEs may involve multiple independent variables, such as time and space. Although

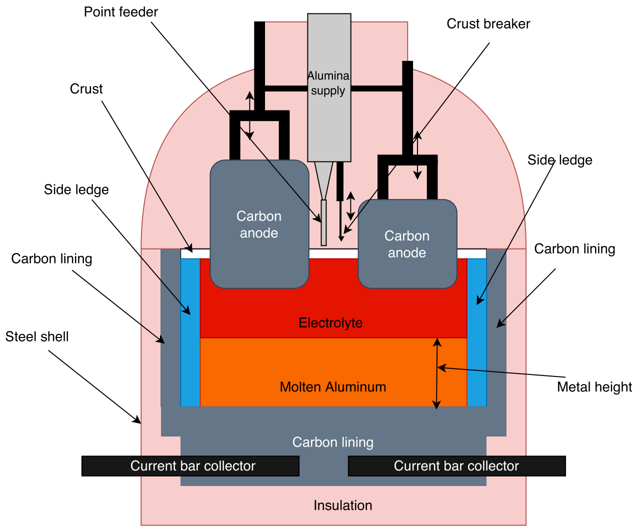


FIGURE 1. Schematic of the aluminum process.

the approaches presented in this work can be applied to both ODEs and PDEs, the focus will be mainly on ODEs.

A general system of ODEs can be written as follows:

$$\begin{aligned}\dot{x}_1 &= f_1(x_1, \ldots, x_n)\\ &\vdots\\ \dot{x}_n &= f_n(x_1, \ldots, x_n),\end{aligned} \tag{1}$$

as provided by [29]. Here, $\dot{x}_i$ denotes the time derivative of x, $\frac{dx_i}{dt}$, and the variables x_i represent the different states of the system, such as temperature, speed, or concentrations of liquids in a chemical process. The functions $f_1, \ldots, f_n$ describe the dynamics of the system by defining how the different states evolve based on the values of all the states in the system.

1) ALUMINIUM EXTRACTION PROCESS

The data in this work come from a simulation of a dynamical system that describes an aluminum extraction process [30], which will be presented as a set of differential equations in state-space form. A schematic of the process is shown in Fig. 1. The model is based on a single aluminum electrolysis cell and is derived using energy and mass balance. Each constant, state, and input is denoted by k_i, x_i, and u_i, respectively. With five inputs, $\boldsymbol{u} \in \mathbb{R}^5$, and eight states, $\boldsymbol{x} \in \mathbb{R}^8$, the model can be written as a set of nonlinear ODEs:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}), \tag{2}$$

where $\dot{\boldsymbol{x}} \in \mathbb{R}^8$ is the time derivative of states $\boldsymbol{x}$, and f is some non-linear function of states and inputs.

The nonlinear dynamics of the system are complex. To make the final equations easier to understand, some equations that partially describe the nonlinear dynamics are presented here:

$$g_1 = 991.2 + 112\, c_{x_3} + 61\, c_{x_3}^{1.5} - 3265.5\, c_{x_3}^{2.2} - \frac{793 c_{x_2}}{-23\, c_{x_2} c_{x_3} - 17\, c_{x_3}^2 + 9.36\, c_{x_3} + 1} \tag{3a}$$

$$g_2 = \exp\left(2.496 - \frac{2068.4}{273 + x_6} - 2.07 c_{x_2}\right) \tag{3b}$$

$$g_3 = 0.531 + 3.06 \cdot 10^{-18} u_1^3 - 2.51 \cdot 10^{-12} u_1^2 + 6.96 \cdot 10^{-7} u_1 - \frac{14.37(c_{x_2} - c_{x2,crit}) - 0.431}{735.3(c_{x_2} - c_{x2,crit}) + 1} \tag{3c}$$

$$g_4 = \frac{0.5517 + 3.8168 \cdot 10^{-6} u_2}{1 + 8.271 \cdot 10^{-6} u_2} \tag{3d}$$

$$g_5 = \frac{3.8168 \cdot 10^{-6} g_3 g_4 u_2}{g_2(1 - g_3)}. \tag{3e}$$

In the equations above, g_1 is the liquidus temperature, T_{liq}, g_2 is the electrical conductivity, κ, g_3 is the bubble coverage, g_4 is the bubble thickness, d_{bub}, and g_5 is the bubble voltage drop, U_{bub}. c_{x_2} and $c_{x_2,\text{crit}}$ are the mass ratio of alumina (Al_2O_3) and its corresponding critical value, respectively. c_{x_3} is the mass ratio of aluminum fluoride (AlF_3). They are, again, given as:

$$\begin{aligned} c_{x_2} &= x_2/(x_2 + x_3 + x_4) \\ c_{x_3} &= x_3/(x_2 + x_3 + x_4). \end{aligned} \tag{4}$$

With the above equations in mind, the state-space equation describing the aluminum electrolysis cell is given by:

$$\dot{x}_1 = \frac{k_1(g_1 - x_7)}{x_1 k_0} - k_2(x_6 - g_1) \tag{5a}$$

$$\dot{x}_2 = u_1 - k_3 u_2 \tag{5b}$$

$$\dot{x}_3 = u_3 - k_4 u_1 \tag{5c}$$

$$\dot{x}_4 = -\left(\frac{k_1(g_1 - x_7)}{x_1 k_0} - k_2(x_6 - g_1)\right) + k_5 u_1 \tag{5d}$$

$$\dot{x}_5 = k_6 u_2 - u_4 \tag{5e}$$

$$\dot{x}_6 = \frac{\alpha}{x_2 + x_3 + x_4}\left[u_2\left(g_5 + \frac{u_2 u_5}{2620 g_2}\right) - k_9 \frac{x_6 - x_7}{k_{10} + k_{11} k_0 x_1} - \left(k_7 (x_6 - g_1)^2 - k_8 \frac{(x_6 - g_1)(g_1 - x_7)}{k_0 x_1}\right)\right] \tag{5f}$$

$$\dot{x}_7 = \frac{\beta}{x_1}\left[-\left(k_{12}(x_6 - g_1)(g_1 - x_7) - k_{13} \frac{(g_1 - x_7)^2}{k_0 x_1}\right) + \frac{k_9(g_1 - x_7)}{k_{15} k_0 x_1} - \frac{x_7 - x_8}{k_{14} + k_{15} k_0 x_1}\right] \tag{5g}$$

$$\dot{x}_8 = k_{17} k_9 \left(\frac{x_7 - x_8}{k_{14} + k_{15} k_0 \cdot x_1} - \frac{x_8 - k_{16}}{k_{14} + k_{18}}\right) \tag{5h}$$

Each of the above states represents some physical quantity, see Table 1 for how they can be interpreted.

TABLE 1. Physical meaning of the variables in the aluminum extraction process, states $x_1, \ldots, x_8$ and inputs $u_1, \ldots, u_5$, along with its units.

Variable	Physical meaning	Unit
x_1	Mass side ledge	kg
x_2	Mass Al_2O_3	kg
x_3	Mass AlF_3	kg
x_4	Mass Na_3AlF_6	kg
x_5	Mass metal	kg
x_6	Temperature bath	$^\circ C$
x_7	Temperature side ledge	$^\circ C$
x_8	Temperature wall	$^\circ C$
u_1	Al_2O_3 feed	kg
u_2	Line current	kA
u_3	AlF_3 feed	kg
u_4	Metal tapping	kg
u_5	Anode-cathode distance	cm

B. NEURAL NETWORKS AND SPARSITY

Neural networks, which have found applications across various scientific disciplines due to their robust modeling capabilities, are structured as interconnected layers of neurons. Their training relies on the backpropagation technique, optimizing parameters through gradient-based optimization of specific loss functions [31], [32]. These networks excel in handling complex and highly nonlinear data, thanks in part to their efficient nonlinear activation functions [33], [34]. In this work, our focus is on fully connected neural network models. These models, based on a comprehensive architecture, interconnect every unit of one layer with every unit of the subsequent layer, with the exception of the input and output layers [31]. This architecture results in a significant number of parameters, including weights for neuron connections and bias terms, excluding input units. The performance of these models depends heavily on the chosen architecture and the training process, often benefiting from access to substantial datasets of high quality. It is worth noting that modern neural networks are often characterized by a high degree of overparameterization [35]. However, techniques such as pruning can effectively reduce network parameters without compromising accuracy and performance. As a result, the concept of introducing sparsity into deep neural networks has gained attention, primarily to optimize memory usage and computational resources. Remarkably, inducing sparsity often has minimal impact on overall performance and generalizability, and in some cases, it can even enhance these metrics [22] while enhancing interpretability. The following subsection introduces the sparsification methods used in this work in greater detail.

1) DYNAMIC SPARSE REPARAMETERIZATION

Methods for iterative pruning and regrowth of weights have existed for some time. However, such techniques often suffer from high computational costs and the need for manual configurations of the number of free weights for each layer.

Dynamic sparse reparameterization (DSR) aims to exploit the benefits of pruning and regrowing without the computational overhead of earlier techniques [21]. The DSR algorithm is a training process that explores the different possible structures of the network during training and aims to identify important structures rather than overparameterizing the network from the start.

DSR aims to introduce a reparameterization of an existing network architecture, rather than designing a new one. A network or a single layer within a network can be represented as $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ denotes the parameters of the network or layer. A general reparameterization is to reparameterize $\boldsymbol{\theta}$ as $\boldsymbol{\theta} = g(\boldsymbol{\phi}, \boldsymbol{\psi})$ with $\boldsymbol{\phi} \in \boldsymbol{\Phi}$, $\boldsymbol{\psi} \in \boldsymbol{\Psi}$, and g being differentiable with respect to $\boldsymbol{\phi}$ but not necessarily with respect to $\boldsymbol{\psi}$. $\boldsymbol{\psi}$ is referred to as a metaparameter, which denotes the parameters of the reparameterization function g and is not optimized by gradient descent, unlike $\boldsymbol{\theta}$. Following this, the general reparameterization can be written as:

$$\boldsymbol{y} = f(\boldsymbol{x}; g(\boldsymbol{\phi}, \boldsymbol{\psi})) \triangleq f_{\psi}(\boldsymbol{x}; \boldsymbol{\phi}). \tag{6}$$

The network can still be trained using backpropagation through g, which is differentiable with respect to $\boldsymbol{\phi}$: $\frac{\partial}{\partial \boldsymbol{\phi}} = \frac{\partial g}{\partial \boldsymbol{\phi}} \frac{\partial}{\partial g}$. With this parameterization, by choosing $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ such that $\dim(\boldsymbol{\Phi}) + \dim(\boldsymbol{\Psi}) < \dim(\boldsymbol{\Theta})$ and $f_{\psi} \approx f$ in terms of generalization performance, f_{ψ} will be a more parameter-efficient function approximation than f.

DSR reparameterizes the network by letting $\boldsymbol{\phi}$ be the values of the non-zero weights in the network and $\boldsymbol{\psi}$ be the indices of the weights in the different layers of the network. The reparameterization function $g(\boldsymbol{\phi}, \boldsymbol{\psi})$ is a placement function that places the weights in $\boldsymbol{\phi}$ at the indexes given in $\boldsymbol{\psi}$.

The reparameterization is applied to the layers of the network. Let all reparameterized layers in the network be denoted as $\boldsymbol{W}_l$, where $l = 1, \ldots, L$, such that $\boldsymbol{W}_l = g(\boldsymbol{\phi}_l, \boldsymbol{\psi}_l)$. The number of zero elements in $\boldsymbol{W}_l$ is denoted as N_l and the non-zero elements of $\boldsymbol{W}_l$ as M_l. Then $N = \sum_l N_l$ is the total number of non-zero parameters in the network and $M = \sum_l M_l$ is the total number of zero parameters in the network.

DSR consists of two phases: training a sparse network for a given number of epoch iterations and redistributing the parameters in the network through pruning and regrowth. The algorithm aims to maintain a given sparsity level S_l throughout training and before and after the pruning/regrowing phase. To do this, define a global desired number of parameters to prune/regrow at each iteration, N_p, and adjust the pruning threshold based on whether this number is met or not. Overall, the algorithm can be summarized in these steps:

1) Train the model parameterized by $\left\{\left(\boldsymbol{\phi}_l^{(t)}, \boldsymbol{\psi}_l^{(t)}\right)\right\}$ for P batch iterations.
2) Prune and regrow the weights of the model according to the DSR algorithm to arrive at $\left\{\left(\boldsymbol{\phi}_l^{(t+1)}, \boldsymbol{\psi}_l^{(t+1)}\right)\right\}$.

The pruning and regrowing phase is the most unique aspect of DSR and includes the following key features.

1) Prune the network, layer-wise, by threshold H. Store the number of pruned parameters globally in variable K.
2) Check if the number of pruned parameters is in the region of the desired number of pruned parameters, N_p. If so, keep H. If not so, adjust H accordingly.
3) Regrow, layer-wise, roughly K parameters. The regrown parameters are distributed across all layers. The heuristic guiding the layer-wise growth is:

$$G_l^{(t)} = \left[\frac{R_l^{(t)}}{\sum_l R_l^{(t)}} \sum_l K_l^{(t)} \right], \tag{7}$$

where $K_l^{(t)}$ is the number of pruned parameters for each layer and $R_l^{(t)} = M_l^{(t)} - K_l^{(t)}$ is the number of surviving weights in a layer.

The entire algorithm can be defined using a few hyperparameters: $(S_l, P, N_p, \delta, H^{(0)})$. S_l is the sparsity level, P is the frequency of reallocation, N_p is the desired number of reallocations, δ is a parameter that affects how strict N_p should be, and $H^{(0)}$ is the initial threshold value.

2) SOFT THRESHOLD REPARAMETERIZATION FOR LEARNABLE SPARSITY

Soft Thresholding Reparameterization (STR) reparameterization for learnable sparsity introduces a new approach to thresholding the weights of neural networks using a learnable threshold parameter [20]. The threshold parameter, s, is the input to a thresholding function $t(s)$ that serves as a thresholding value for the weights in the network. The thresholding function is incorporated into the loss function in the network, allowing it to be optimized during training.

The concept of soft thresholding is not new and was introduced in [36] in 1995. The soft thresholding function is applied to the weights of each layer to create the soft thresholded version of the weights $\mathbf{W}_l$ of the l-th layer in the network: $S(\mathbf{W}_l, \alpha_l) := \text{sign}(\mathbf{W}_l) \cdot \text{ReLU}(|\mathbf{W}_l| - \alpha_l)$, where α_l is the pruning threshold for the l-th layer. Because the loss function can be written as a continuous function of the α_l's, the backpropagation algorithm is still applicable for learning both the values of the weights and the threshold α_l. This results in a layer-specific and learnable soft thresholding technique.

Since each layer in a neural network provides different information extraction features to the network, it is interesting that it is possible to learn unique threshold values and sparsity for each layer. This allows for fine-tuning of the network's sparsity and pruning behavior, which can improve the network's performance.

The reparameterization part of this approach is to reparameterize the weights of the network as $S_t(\mathbf{W}; s)$ and, instead of updating the weight matrix first, directly optimize the projection of the weights to find the optimal combination of weights and sparsity level. The reparameterization of $\mathbf{W}$, $S_t(\mathbf{W}; s)$, is parameterized by the threshold parameter s and the thresholding function t. The projection is applied element-wise to the weight matrix as follows:

$$S_t(w, s) := \text{sign}(w) \cdot \text{ReLU}(|w| - t(s)). \tag{8}$$

Note that $\text{ReLU}(a) = \max(a, 0)$, hence if $|w| \leq t(s)$, then $S_t(w, s) = 0$. The introduction of STR modifies the parameter optimization problem of neural networks to:

$$\min_{\mathcal{W}} \mathcal{L}(S_g(\mathcal{W}, s), \mathcal{D}). \tag{9}$$

A typical DNN architecture is divided into L-layers. Something that makes is possible to also divide $\mathcal{W}$ into L different weight matrices: $[\mathbf{W}_l]_{l=1}^{L}$. This opens the possibility for assigning an individual learnable weight parameter s_l to each layer resulting in $\boldsymbol{s} = [s_1, \ldots, s_L]$.

One challenge of using STR is that it is difficult to set an explicit overall sparsity budget. Instead, the sparsity level is determined by the hyperparameters, primarily the type of regularization applied and its parameter value, but also by adjusting the initial values of the sparsity parameter s_{init}. Typically, to avoid pruning all small weights in the beginning, s_{init} is set to values that make $t(s)$ close to zero.

3) MAGNITUDE PRUNING

Conventional neural networks fix the architecture before training, so the architecture cannot change or improve during training. To address this, magnitude pruning tries to learn the important weights and connections by iteratively removing a fraction of the smallest weights in each layer. This allows the network's architecture to adapt and improve during training [19].

The magnitude pruning algorithm is initialized with a fully-connected neural network with a sparsity of 0. The user specifies the total number of epochs to train N, when to prune M, and the desired sparsity s. The network is then trained for M iterations before pruning. The pruning step involves calculating a layer-specific weight threshold by sorting the weights and finding the threshold that yields the desired sparsity. After pruning, the remaining non-zero weights are trained for $N - M$ iterations.

By repeating this training and pruning process several times with different learning rates, number of epochs, and sparsity levels, it is possible to find sparse representations of the network that perform as well as fully dense networks in terms of generalizability and fit to the data.

Traditionally, magnitude pruning has been used to reduce the memory usage and computational cost of large neural networks (e.g. convolutional networks for image recognition). For example, in [19] the authors managed to reduce the number of parameters from 138 million to 10.3 million without any loss of accuracy using this approach.

4) L_1-REGULARIZATION

L1-regularization, also known as the Least Absolute Shrinkage and Selection Operator (LASSO) regularization, is a widely used regularization technique in the fields of machine learning and statistics. It works by adding an extra term to the model's loss function, which consists of a predetermined scaling factor and the sum of the absolute values of the weights. This term penalizes the weights with respect to their absolute values, hence encouraging them to take on smaller values, reducing overfitting and improving the model's generalizability. This extra term consists of a predetermined scaling factor and the sum of the absolute value of the weights:

$$L_1 = \lambda_1 \cdot \sum_{i=1}^{N} \sum_{j=1}^{M} |w_{i,j}|, \tag{10}$$

where λ_1 is the L_1 scaling factor and N and M are the numbers of rows and columns of the weight matrix respectively. Lastly, $|w_{i,j}|$ is the absolute value of the weight at position (i, j) in the weight matrix.

By encouraging small weights to take on values of zero, L1-regularization leads to sparser networks less prone to overfitting. Similarly, by keeping the weights with higher magnitudes the network retains the significant parts and only the less important are removed. This can improve the model's generalizability and reduce overfitting.

Another commonly used regularization method is L2-regularization, which uses the squared values of the weights as a penalty, instead of the absolute values. L2-regularization punishes higher-valued weights more harshly than lower-valued weights, while L1-regularization punishes all weights equally. When training a neural network using L2-regularization, the values of the weights will tend towards zero, but they will not reach as small values as they do with L1. This is because the gradient of the penalty term for the weights becomes increasingly small as the weights approach zero. In contrast, if the networks are trained using L1-regularization, more weights will become even closer to zero because the gradient of the penalty term is always 1 for positive weights and -1 for negative weights, regardless of their magnitude. L1-regularization will rarely force the weight to exactly 0, but their value often becomes so small that they may be regarded as 0, which is more interesting for sparsification.

C. GENERALIZATION CAPABILITY OF NEURAL NETWORKS

Generalizability is a fundamental aspect of the design and evaluation of neural network models. In brief, the ability of a model to generalize refers to its capacity to make accurate predictions on previously unseen data. This property is desirable in neural network models because it allows them to effectively capture the underlying patterns in a dataset and make reliable predictions on new unseen instances. This is especially crucial in real-world applications where the performance of a model is judged based on its ability to accurately make predictions on novel data.

In the context of using neural networks to simulate dynamical systems, generalizability is particularly relevant. This is because the ability of a neural network to accurately capture the underlying dynamics of a system and make reliable predictions on unseen data is essential for its performance. For instance, when simulating the behavior of a physical system, a neural network with strong generalizability will be able to accurately predict the future state of the system based on its current state and the external forces acting on it. A model with good generalizability will also be able to handle different initial conditions of the system.

1) STOCHASTIC WEIGHT AVERAGING

Stochastic Weight Averaging (SWA) is a method that aims to improve the generalizability of the final model by averaging the model weights during parts of the training procedure. This technique has been shown to improve the performance of neural networks trained using SGD-based optimizers by closing in on the optimal solution of the optimization problem, with almost no additional computational overhead [37].

The SWA algorithm begins by using a set of pre-trained weights, denoted $\hat{w}$, as a starting point. These weights may be the result of a complete or partial network training procedure. The training process then continues using either a cyclic or a constant learning rate. Unlike normal training steps, the weights are now a function of both the new weights updated using stochastic gradient descent (SGD) and the prior weights. Upon completion of this training procedure, the final weights w_{SWA} are obtained.

Algorithm 1 Stochastic Weight Averaging

```
Require: weights $\hat{w}$, number of iterations $n$, learning rate
    $\alpha$, loss function $\mathcal{L}$
Ensure: $w_{\text{SWA}}$
  $w \leftarrow \hat{w}$ {Initialize weights with $\hat{w}$}
  $\hat{w}_{\text{SWA}} \leftarrow w$
  for $i \leftarrow 1, 2, \dots, n$ do
    $w \leftarrow w - \alpha \nabla \mathcal{L}(w)$ {Stochastic gradient update}
    $w_{\text{SWA}} \leftarrow \frac{w_{\text{SWA}} \cdot n_{\text{models}} + w}{n_{\text{models}} + 1}$, {Update average}
  end for
```

In the average function:

$$w_{\text{SWA}} \leftarrow \frac{w_{\text{SWA}} \cdot n_{\text{models}} + w}{n_{\text{models}} + 1}, \tag{11}$$

w_{SWA} is the SWA version of the weights, n_{models} is the number of models used in the weight averaging procedure, and w is the version of the weight from the current gradient descent update. When the above calculation is the only extra step when using this algorithm compared to normal weight update, it is clear that this technique does not introduce much overhead.

SWA is closely related to model ensembling, which is a group of techniques that combine the outputs of multiple models to generate a more accurate prediction or output. For example, the Fast Geometric Ensambling method proposed in [38] uses the average of the model outputs to improve performance. In contrast, SWA uses the average of the weights to generate better results. Both approaches aim to produce an output that is as close as possible to the optimal model.

To better understand why the SWA method may be effective in training a neural network, let's take a closer look at the underlying mathematics. Start by considering the dimensionality of the weight space of a neural network, denoted by d. For each iteration of Stochastic Gradient Descent (SGD), a new set of weights, w_i, is found, where $i = 1, 2, \dots, k$ and k is the total number of iterations. Assume that all of the weights, w_i, are located close to some local optimum $\hat{w}$. SWA then calculates the average of the gradient-updated weights at all time steps, resulting in $w_{\text{SWA}} = \frac{1}{k}\sum_{i=1}^{k} w_i$. Under certain assumptions, it can be shown that the weights for all iterations of the gradient update are samples from a multidimensional Gaussian distribution, $\mathcal{N}(\hat{w}, \Sigma)$, where Σ is a covariance matrix defined by the optimization surface, batch size, and learning rate. In a d-dimensional space, the samples of a multidimensional Gaussian distribution are concentrated in an ellipsoid:

$$\left\{ z \in \mathbb{R}^d \mid \|\Sigma^{-\frac{1}{2}}(z - \hat{w})\| = \sqrt{d} \right\}. \tag{12}$$

In a multidimensional setup, the probability of an updated weight sample, w_i, ending up close enough to $\hat{w}$ is negligible. However, for SWA, by sampling from this distribution, w_{SWA} is guaranteed to converge to $\hat{w}$ as k approaches infinity [37].

Intuitively, by considering a set of points all with relatively low training loss, running SGD in this area will result in the algorithm iterating over the surface of this set. With enough iterations, it becomes clear that averaging the weights will converge to a more general solution. Most likely, averaging over different low-valued solutions will move w_{SWA} to a more central point that is closer to the optimal solution.

D. NEURAL NETWORKS AND DYNAMICAL SYSTEMS

1) NEURAL NETWORKS IN THE CONTEXT OF DYNAMICAL SYSTEMS

One potential application of neural networks, and the main focus of this work, is in the simulation of dynamical systems. This use of neural networks was first proposed in 1990 [11]. As a dynamical system is a system that evolves over time according to a set of rules or equations, neural networks can be used to simulate such a system by learning these underlying rules or equations that govern its behavior. Many dynamical systems exhibit highly nonlinear and chaotic behavior, for which analytical solutions do not exist, and obtaining numerical solutions is computationally demanding [29]. In these cases, neural networks can be a valuable tool for improving the accuracy of simulations and gaining a deeper understanding of the behavior of the system.

However, there are also potential limitations to using neural networks for simulating dynamical systems. Neural networks require a large amount of data during training, which may not always be available. Additionally, the quality of the data can greatly affect the performance of a neural network [31].

In many real-world applications of neural networks and dynamical systems, the data used to simulate or predict the states of a system may be sparse and/or noisy, which can make training challenging. Another challenge of using neural networks in this context is the black-box nature of these models. While they are effective at approximating nonlinear functions and complex dynamics, there is no guarantee of stability or performance.

There are various approaches for using neural networks to simulate dynamical systems and the reader can refer to [12] for an overview of some common approaches. In all of these approaches, a neural network is trained and then used as part of the prediction process. The simulation schemes can broadly be divided into two categories: direct-solution models and time-stepper models. A direct-solution model maps a given time, t_k, to a solution at that timestep, x_k. A time-stepper model, on the other hand, approximates the derivative of the system in some way before integrating it over time.

A simple example of a direct-solution model is a vanilla direct-solution model. Such a model estimates the function x_k at timestep t_k by learning how a function evolves over time. Vanilla direct-solution models typically perform well when given training data of good quality. However, if the quality of the training data is poor or the model is presented with samples outside of its training data (extrapolation), it usually performs poorly. Another, more complex, version of a direct-solution model is a physics-informed neural network (PINN) [39]. As the name suggests, a PINN is a type of neural network in which knowledge about the system being simulated is incorporated into the network, often as an additional term in the loss function.

Time-stepper models are more similar to traditional numerical solvers as the derivative of the model is approximated at each time step, t_k, and the model is integrated through time. A benefit of such models is that they can make use of knowledge and be combined with classical integration schemes. An *Euler time-stepper* is used to simulate the system in this work. The Euler time-stepper tries to learn the derivative of the system at a timestep, t_k, given the estimated states of the system. Starting with a given set of initial conditions the method estimates the derivative at each timestep given the internal states and inputs to the system. Mathematically, it integrates through time by

$$\begin{aligned} \dot{\tilde{\boldsymbol{x}}}_t &= f(\tilde{\boldsymbol{x}}_t; \boldsymbol{\theta}) \\ \tilde{\boldsymbol{x}}_{t+1} &= \tilde{\boldsymbol{x}}_t + \Delta T \cdot \dot{\tilde{\boldsymbol{x}}}_t, \end{aligned} \tag{13}$$

where $\dot{\tilde{\boldsymbol{x}}}_t$ is the derivative of the system at the current timestep estimated by the neural network. $f(\tilde{\boldsymbol{x}}_t; \boldsymbol{\theta})$ is a neural network with parameters $\boldsymbol{\theta}$ taking prior estimated values of the states at the current input, $\tilde{\boldsymbol{x}}_t$ as input. $\tilde{\boldsymbol{x}}_{t+1}$ is the estimated value of the states at the next timestep, and it is found by taking the current state value and adding the estimate of the derivative time the size of the timestep, ΔT. It is worth noting that the inputs to a system may be known and their exact value can be incorporated into $\tilde{\boldsymbol{x}}_t$.

How to use the Euler time-stepper to estimate the values of the dynamical system through the time horizon is shown in Algorithm 2.

Algorithm 2 Neural Network Based Euler Time-Stepper

```
Require: Initial conditions, $\boldsymbol{x}_0$, size of timestep $\Delta T$,
    prediction horizon, $T$, retrained neural network, $f$, with
    parameters $\boldsymbol{\theta}$.
    Initialize prediction vector, $\boldsymbol{x}_{\text{pred}}$
    Insert initial conditions as first input in prediction
    vector: $\boldsymbol{x}_{\text{pred}}[0] \leftarrow \boldsymbol{x}_0$
    Add prior information to $\boldsymbol{x}_{\text{pred}}$ such as controllable
    inputs to the system.
    for i = 1, ..., $T$ do
        Predict derivative: $\dot{\boldsymbol{x}} \leftarrow f(\boldsymbol{x}_{\text{pred}}[i]; \boldsymbol{\theta})$
        Update next prediction step, $i + 1$:
        $\boldsymbol{x}_{\text{pred}}[i+1] \leftarrow \Delta T \cdot \dot{\boldsymbol{x}}$
    end for
```

2) INDUCING SPARSITY TO IMPROVE STABILITY, GENERALIZABILITY, AND ACCURACY IN SIMULATIONS OF DYNAMICAL SYSTEMS USING NEURAL NETWORKS

As previously mentioned, neural networks are often highly overparameterized, with many state-of-the-art models in image classification and natural language processing having millions or even billions of parameters [40], [41]. This can lead to computationally demanding training and inference processes. Sparsity techniques aim to address this issue by removing small and less important weights and connections from the model without significantly reducing its performance. If successful, this means that the patterns and information in the data are primarily described by a small subset of the network's weights and connections.

The theory that inducing sparsity into neural networks used for the simulation of dynamical systems may lead to more stable, generalizable, and accurate models is based on the following premise. Given a dense, fully connected neural network trained on data from some dynamical system, provided that the network is large enough, it is likely capable of learning the dynamics of the system to some extent. However, when trying to use the network to make predictions using initial conditions or time horizons outside of what it saw in the training data, it will likely fail. This indicates that, although the model is able to interpolate well, it has not truly "learned" the general dynamics of the system, but rather has simply memorized how the dynamical system behaves within a limited horizon and set of initial conditions. The goal of inducing sparsity in such a model is to force the model to learn connections that represent the actual dynamics, rather than simply memorizing the data. This can lead to models that are more stable and that generalize better on unseen data. Ultimately, this approach may even enable the use of neural networks to discover unknown dynamics or physical equations.

All of the sparsity techniques mentioned above have been shown to result in a significant reduction in the number of parameters without reducing the accuracy of the model. Therefore, it is believed that they may also apply to smaller networks used for simulating dynamical systems. However, it is important to keep in mind that there is a significant difference between large image classification models and small, fully-connected neural networks. This may affect the performance of the sparsified networks in the context of simulating dynamical systems.

III. METHOD

A. DATASET

Numerous dynamical systems can be found online with code examples to generate data [42], [43]. These systems offer various scenarios to test model performance. However, for real-world applications, working with actual asset data is more compelling. Therefore, we use data from a real aluminum extraction process to assess our models. Ideally, real-data measurements would train models and validate predictions against actual asset states. However, obtaining these measurements is costly and time-consuming. The aluminum extraction process presents such challenges, emphasizing the need for accurate predictive models. To this end, we generate data by simulating the aluminum extraction process numerically, as detailed in II-A1. In Fig. 2 five simulations of the internal states from the test data set are shown, all with different initial conditions. The inputs corresponding to the simulations are plotted in Fig. 3. This demonstrates the complexity of the dynamical system and the importance of accurate modeling and prediction.

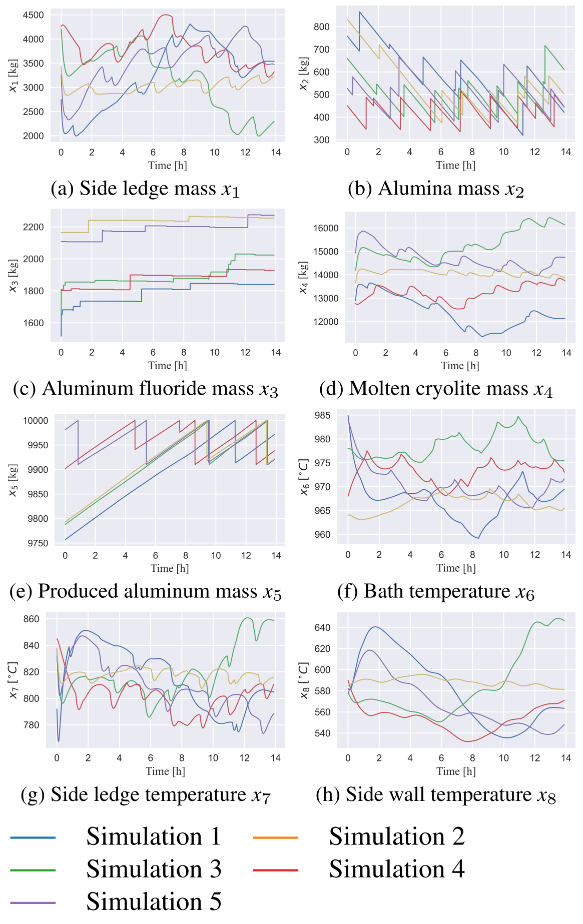


FIGURE 2. The figure shows plots of 5 simulations of the internal states, $x_1, \ldots, x_8$, of the dynamical system describing the aluminum extraction process. The different states exhibit very different dynamics that are challenging to simulate, both with and without the use of neural networks.

B. DATA GENERATION

The training set, the validation set, and the test set each consist of 40 simulations for 5000 timesteps for 13 states. The data were generated using high-fidelity numerical simulation. 40 models were trained for each sparsity technique, with 10 models using the regular training process, 10 using L1 regularization during training, 10 using SWA during training, and 10 using a combination of L1 regularization and SWA. The different simulations and data sets are distinguished by their unique sets of initial conditions and inputs. The initial conditions are randomly sampled from a predefined range of possible values. Since the different states represent different physical quantities, each state has a unique range and initialization process. This allows for a diverse and representative data set that captures the complexity of the underlying dynamical system.

C. DATA PRE-PROCESSING

Learning the dynamics requires advancing the state in time. With a neural network, this will require learning the derivatives. Hence, the derivative at each timestep was estimated using the following equation:

$$\dot{\boldsymbol{x}} = \frac{\boldsymbol{x}_{t+1} - \boldsymbol{x}_t}{\Delta T},$$

where $\dot{\boldsymbol{x}}$ is a vector of the time derivatives of the states, $\boldsymbol{x}_{t+1}$ is a vector with the state values at timestep $t+1$, and $\boldsymbol{x}_t$ is a vector with the state values at timestep t. ΔT is the size of each timestep. This $\dot{\boldsymbol{x}}$ vector served as the target during training.

Another pre-processing step applied to the data was normalization. Normalizing data before using it in the training process of neural networks can significantly reduce both the prediction error and the time it takes to find an optimal solution [44]. The data is normalized according to the following equation:

$$\boldsymbol{x}_{\text{norm}} = \frac{\boldsymbol{x} - \boldsymbol{x}_{\text{mean}}}{\boldsymbol{x}_{\text{std}}},$$

where $\boldsymbol{x}$ is the entire data set, in this case of size $[40, 5000, 13]$, $\boldsymbol{x}_{\text{mean}}$ and $\boldsymbol{x}_{\text{std}}$ are both vectors of size $[1, 13]$. Each element in $\boldsymbol{x}_{\text{mean}}$ and $\boldsymbol{x}_{\text{std}}$ corresponds to the mean and standard deviation of its corresponding state, respectively.

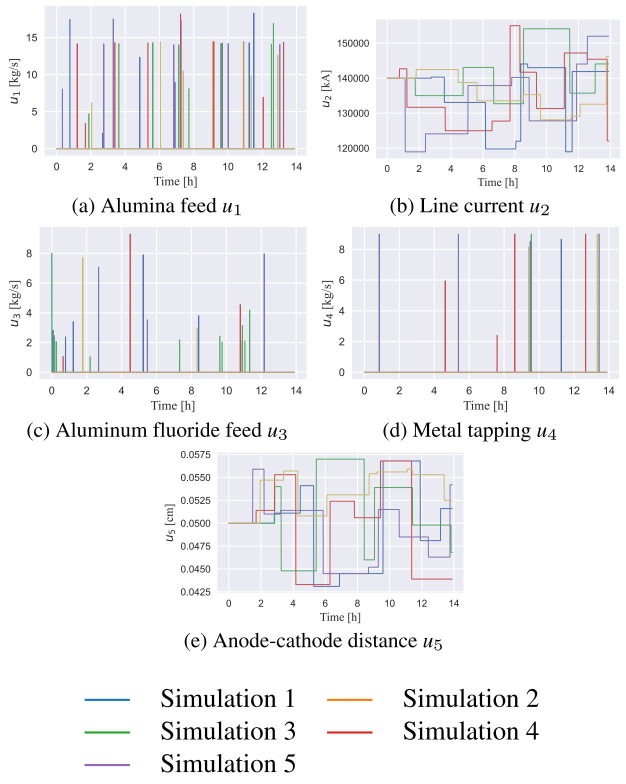


FIGURE 3. The figure shows 5 simulations of the inputs, $u_1, \ldots, u_5$, to the aluminium extraction process. As they are inputs, they are known for all timesteps throughout the simulation.

D. MODEL ARCHITECTURE

All models used in this work are based on fully-connected Multilayer Perceptron neural networks. Most sparsity methods work by performing thresholding, pruning, or regularization during network training. However, some methods require initial sparsity at the start of training. The models have several features in common: they are all trained using the Adam optimizer [45] with a learning rate of 10^{-3} with a scheduler, the cost function is calculated using the MSE loss function, the weights are initialized using the He initialization method [46], and the activation function in all layers (except for the last) is ReLU. All models are also trained using a step-wise learning rate scheduler. Although varying a bit, it was applied such that the learning rate was reduced by 0.1 at each 10-20 epoch. This is the base architecture of all models, and any variances in it are specified in their respective descriptions.

Hyperparameters and layers' architecture were chosen based on the performance of the dense models by hand-tuning. The basic characteristics of the dense models are described in the following subsection. Note the common characteristics that are described before also apply. To optimize the models some architectures were tried out with different numbers of hidden neurons and layers. Models with 2 to 5 hidden layers, each hidden layer with 15, 25, or 40 hidden neurons. The hand-tuning process involved the use of two optimizers, Adam and stochastic gradient descent with and without momentum [47]. Learning rates of 0.1, 0.05, 0.01, 0.005, 0.001, and 0.0001 were tested, and batch sizes of 32, 64, 128, and 256. ReLU was the only activation function used throughout the process, as it typically behaves well for ANN and, particularly, for regression tasks performed by ANN. Note that the purpose of this work is to study sparsity techniques in ANN used to simulate dynamical systems, hence the optimization process is of little interest from the moment the models provide accurate solutions in terms of the metric chosen, MSE in this case.

1) DENSE MODEL

The dense "vanilla" model serves as the baseline model for the experiments and is the simplest model considered. A dense "vanilla" neural network is a fully connected neural network with ReLU activation functions that is trained using the Adam optimizer. "Vanilla" implies that no significant changes are made to the network architecture or training process to improve the network's performance. The network consists of an input layer of size 13 (corresponding to the number of system states), three hidden layers of size 25, and an output layer of size 8 (matching the number of internal states to be estimated). This results in a neural network with dimensions [13, 25, 25, 25, 8].

2) L1-REGULARIZATION MODEL

The L1-regularized model is a dense model trained using L1-regularization. See III-D1 for details about network size. L1-regularization is applied during training by adding an additional term, $L_1 = \lambda_1 \cdot \sum_{i=1}^{N} \sum_{j=1}^{M} |w_{i,j}|$, to the cost function. The value of λ_1 was 10^{-4} for all cases of L1-regularization.

3) SOFT THRESHOLD REPARAMETERIZATION MODEL

As was previously the case with L1-regularization, adding the sparsity technique of soft thresholding for learnable sparsity to a neural network model requires no modifications to the basic network architecture. Thus, the same dense fully connected network as presented in III-D1 serves as a basis for this case. What differs from a dense network, in this case, is the addition of a layer-specific learnable thresholding parameter, s, that needs to be initialized to some value. A simple hyperparameter search was performed by testing different values. This resulted in $s = -2$ being the initial s-value for all models. Also, during the training of this model, the thresholding is performed according to the value of s for the weights of each layer when feeding data through the networks.

4) MAGNITUDE PRUNING MODEL

The process of magnitude pruning is based on the simple idea of removing a specified proportion of the smallest weights in a neural network. To implement this technique functionality for sorting and setting to zero the smallest weights in a layer had to be added. Apart from this added functionality, the magnitude pruned model is the same as the dense model described in III-D1. However, because this model prunes its smallest weights iteratively during training, the final magnitude pruned model has a predefined level of sparsity. To provide additional flexibility, the network size was increased compared to the dense model, to a size of [13, 64, 64, 64, 8].

5) DYNAMIC SPARSE REPARAMETERIZATION MODEL

The dynamic sparse reparameterization (DSR) model is the most different from the dense model described in III-D1. This technique involves changes to both network initialization and training. To implement this technique, the network is initialized with a fraction of the weights set to zero. The DSR model also has specific pruning and regrowing phases that require special algorithms. In order to execute these phases properly, additional functionality was added on top of the dense net. This includes functionality for re-initializing a given number of weights at random zero-valued positions in each layer, and for pruning all weights below an adaptive threshold value, among other things. See II-B1 for details on all algorithms related to the DSR model. Because the DSR model has a set sparsity level after initialization and its performance was poor, it was modified to use a larger network, with dimensions [13, 128, 128, 128, 8].

6) ADDING STOCHASTIC WEIGHT AVERAGING

Stochastic Weight Averaging (SWA) is a technique that can be added during the training of neural networks. SWA simply computes the average of the weights in the network over a subset of the training epochs. In this study, it was added during the last 25% of the training epochs of the different models. As suggested in [37] on SWA, the learning rate was increased and kept constant during the epochs in which SWA was applied.

E. SCENARIO AND CASE SET-UP

The objective of this study is to predict the evolution of the internal states of a dynamical system over time by estimating the derivative of the states. Algorithm 2 illustrates how to use a neural network to estimate the derivative of a system and integrate forward in time based on this estimate. The theory, presented in Section II, provides all of the necessary information to implement the different algorithms, and Section III-D presents the specific details of the implementation and model training.

To evaluate the performance of the sparsity methods and their combinations, ten models are trained and evaluated for each case. The models are then tested by simulating 40 trajectories from a test set according to Algorithm 2. Ten models are trained for each case because their performance may vary based on randomness in their initialization and training. The performance is evaluated by measuring the number of models that diverge and the Rolling-Forecast-Mean-Squared-Error (RFMSE) of the simulations. RFMSE is the mean square error at all timesteps through the simulation, calculated for each state of the model and summarized as the mean of the normalized error for all states. A model is considered to have diverged if the RFMSE of one of its states is more than three times the standard deviation of that state.

IV. RESULTS AND DISCUSSIONS

In this work, sparsity-promoting techniques like L1-regularization, soft thresholding, magnitude pruning, and dynamic sparse reparameterization (DSR) have been applied and studied. The goal was to determine whether these techniques can improve the accuracy of the simulation and reduce the number of simulations that diverge. Stochastic Weight Averaging (SWA), a technique to improve the generalization of neural networks, was also applied.

In Section IV-A, the results of applying sparsity techniques to dense vanilla neural networks are presented. In Section IV-B, L1-regularization is applied to the training process of the different networks with different sparsity techniques. Section IV-C presents the results of using SWA to improve generalization in the training process. Finally, Section IV-D investigates whether combining all of these techniques leads to performance improvements or degradations.

The results are presented in two plots: a bar plot showing the percentage of simulations that diverge at different points throughout the simulation horizon and a violin plot showing the RFMSE at the same points during the simulations. To determine whether or not a simulation has diverged, the RFMSE is used. A model is considered divergent if its RFMSE in one of its states is greater than three times the standard error of the state. The points in both graphs are at $100\Delta T$, $1000\Delta T$, $2500\Delta T$, and $5000\Delta T$, where ΔT is the timestep. The violin plot provides information on the distribution of the RFMSE values, while the bar plot provides a simple visualization of the stability of the simulations. Together with the numerical results, these plots will be used to evaluate the stability and generalizability of the models, which are features that we consider key for dynamical system simulation.

To provide some context regarding different networks' performance and show the trajectories forecasted by the models, the following two simulation examples shown in Fig. 4 and Fig. 5 illustrate the performance of 10 models with L1 regularization and soft thresholding respectively.

A. DENSE NETWORKS

Fig. 6 shows the divergence frequency of the different sparsity techniques at different timesteps. For each sparsity technique, 10 different models were trained using the dense network, and each model was used to predict 40 sets of test data. Table 2 shows the divergence rate percentages of all models. The vanilla dense network performs well, with less than 15% of the models diverging throughout the test horizon. Soft thresholding also performs well for the first 1000 ΔT's, with less than 20% divergence. However, after $1000\Delta T$'s, the divergence rate increases significantly. For the magnitude pruning models, only a few percent of the models diverge at

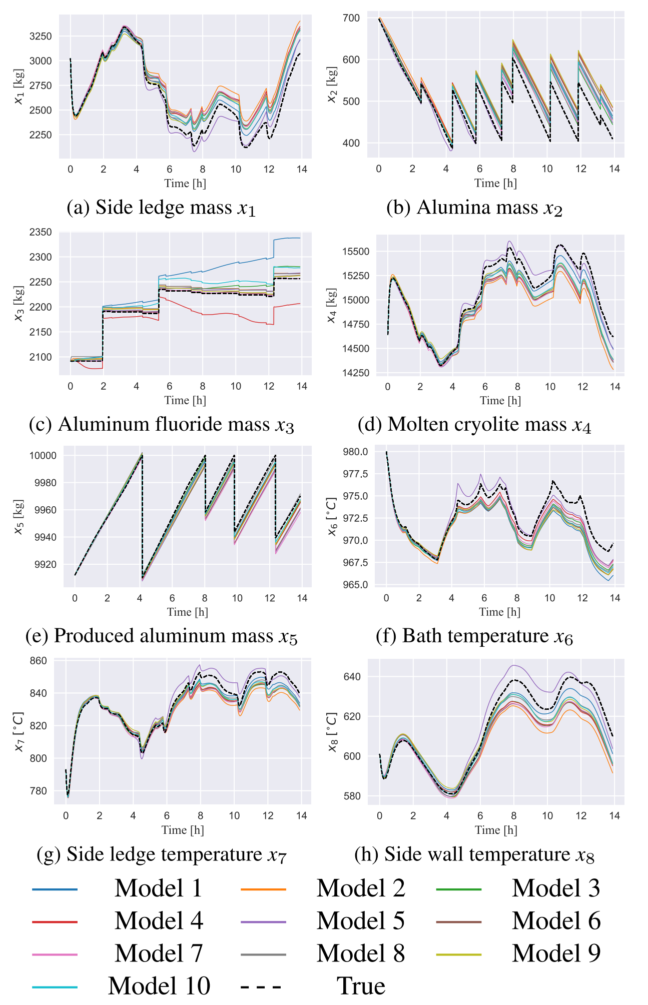


FIGURE 4. **The figure shows the true trajectories of the internal states of the aluminum extraction process along with simulations from 10 different L1-regularized models.**

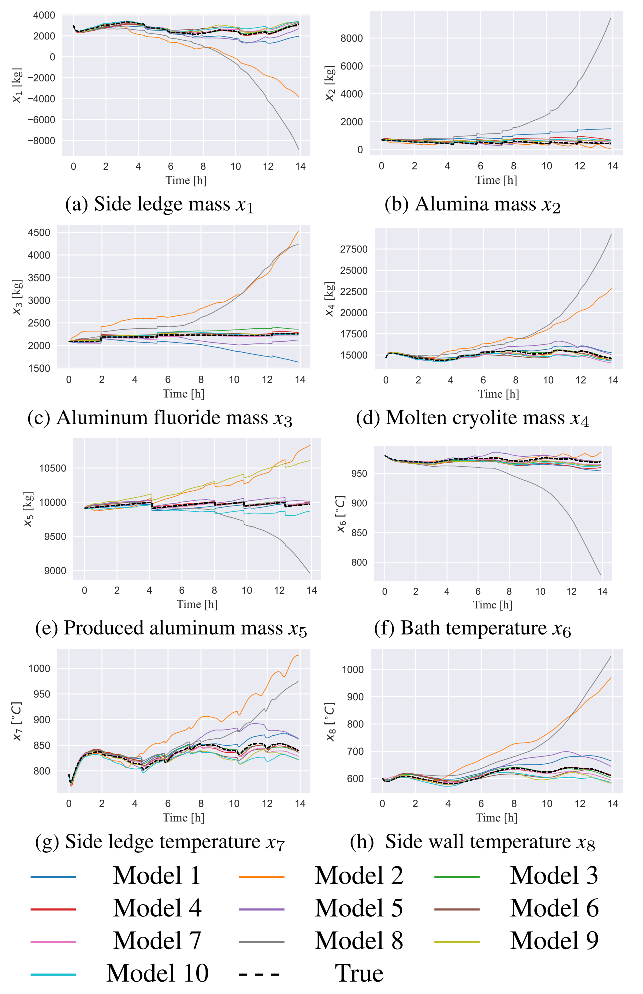


FIGURE 5. **The figure shows the true trajectories of the internal states of the aluminum extraction process along with simulations from 10 different models trained with soft thresholding. The divergence can be observed as time proceeds in the simulation.**

$100\Delta T$. However, after $1000\Delta T$, over 30% of the models diverge. The DSR models perform poorly from the start, with 3.75% of the models diverging at $100\Delta T$, indicating how inadequate these architectures can be for dynamical system simulation with predictions that quickly diverge and get out of control.

The violin plot in Fig. 7 illustrates the RFMSE of non-diverging simulations for various techniques. As the DSR simulations, at one point, has no non-diverging simulations it is not included in this plot and can be considered the worst performing. All model types exhibit an increasing RFMSE throughout the prediction horizon with the exception of the dense model, which exhibits a decrease in its maximum RFMSE from $2500\Delta T$ to $5000\Delta T$. This decrease is likely due to a simulation diverging and therefore being excluded from the calculations. The plot displays the apparent effectiveness of L1-regularization in producing denser distributions of RFMSE, which seem to reduce the increase in RFMSE for the L1 models. There is a discernible pattern in the change of RFMSE for the other models; their distributions begin with most simulations having an RFMSE below the mean, but over the course of the simulations, the densest part of the RFMSE distribution shifts increasingly toward and above the mean.

It is not surprising that applying L1-regularization to a dense model improves the simulations, as training a dense model without any regularization usually results in overfitting. L1-regularization is known to be effective against this and its low divergence rates and dense RFMSE distributions are likely a direct result of the generalization improvement. This also suggests that L1-regularization drives the models to learn the dynamics and reduce data memorization. The soft thresholding model, despite having learnable sparsity, seems to induce too much sparsity in the model, resulting in diverging simulations. However, compared to a sparsity technique with a predefined sparsity, such as the magnitude pruning models, it appears to improve performance by inducing sparsity at the right weights in the right layer of the network. When used without any regularization technique, the magnitude pruning model seems to remove too many weights, indicating that the models may not be able to represent important dynamics in their layers, but rather spread

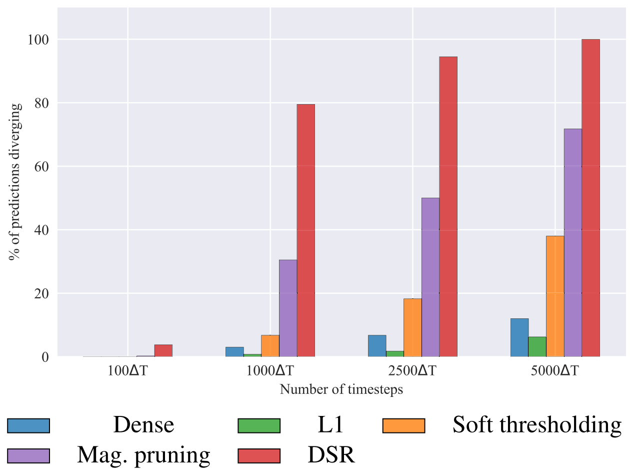


FIGURE 6. The figure presents a bar plot of the percentage of diverging simulations for models that combine a dense network with various sparsification methods. The models that employ the "vanilla" dense approach and the L1-regularized approach exhibit a low percentage of diverging simulations, while those trained with magnitude pruning and DSR have a high percentage of divergence.

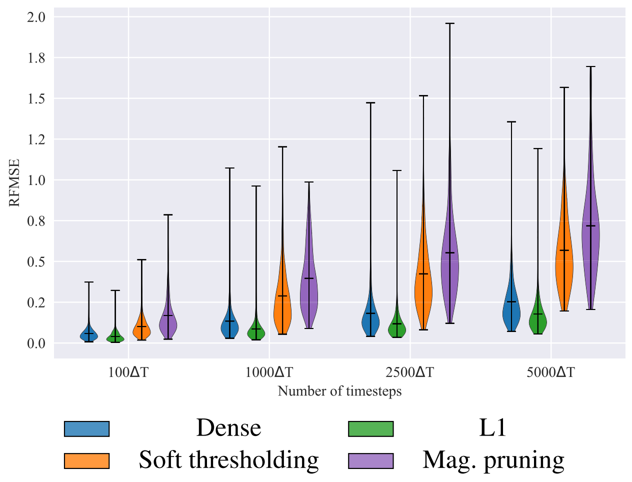


FIGURE 7. The figure displays a violin plot of the RFMSE of the non-diverging simulations for various models. The dense model and the L1-regularized model have low RFMSE and compact error distributions, particularly at early time steps, while the models trained with soft thresholding and magnitude pruning exhibit higher RFMSE and less compact error distributions. The simulations by models trained with DSR are not included in the plot because they had no non-diverging models at some timesteps.

information across many low-valued weights. Removing these weights leads to poor performance. For DSR, it seems that pruning and regrowing destroy many of the initial connections made at the start of training, and the networks are unable to relearn these connections. From the results displayed in Fig. 6 and Fig. 7, it is easy to recognize that the best-performing models are the L1-regularized, with less divergence and better RFMSE.

B. L1-REGULARIZATION

In the experiment results shown in Fig. 8 and Table 2, L1-regularization was added to the training of all model types. As mentioned earlier, the results show that the number of divergent simulations decreased significantly for dense

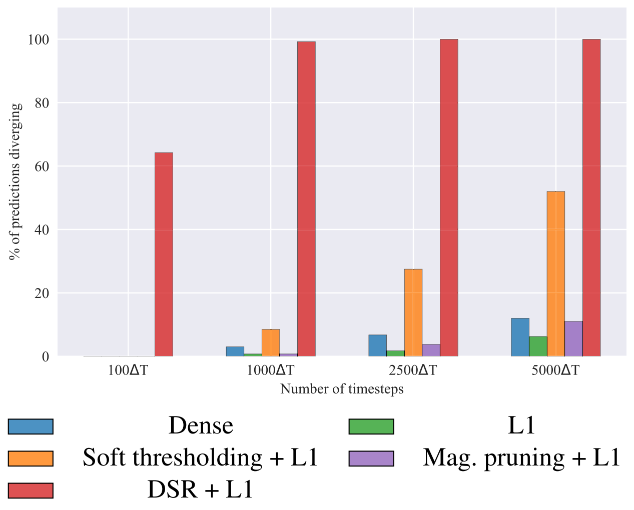


FIGURE 8. The figure presents a bar plot of the percentage of diverging simulations for models that combine L1-regularization with different sparsification methods. The "vanilla" dense model is also included in the plot. The models with L1-regularization only and those that combine L1-regularization with magnitude pruning exhibit a low percentage of divergence, while the models that combine L1-regularization with DSR show a high percentage of divergence.

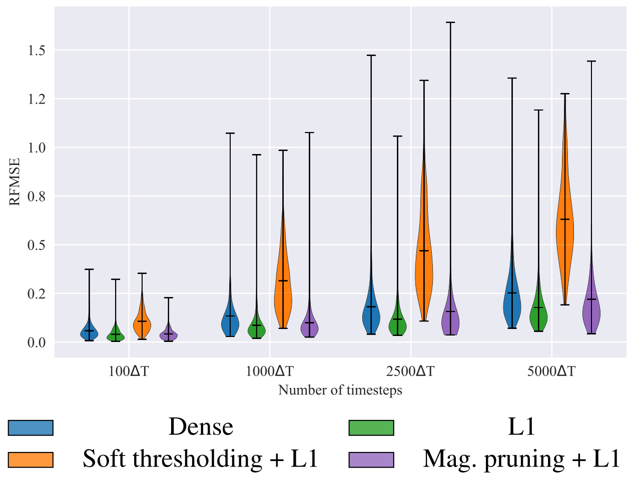


FIGURE 9. The figure shows a violin plot of the RFMSE of the simulations for models that combine L1-regularization with different sparsification methods. The "vanilla" dense model is also included for reference. The models with L1-regularization and the models with L1-regularization plus magnitude pruning have significantly denser error distributions and lower maximum errors than the others, especially at early timesteps. The simulations by models trained with DSR are not included in the plot because they had no non-diverging models at some timesteps.

models. However, the number of diverging simulations also decreased significantly for magnitude pruning models, with a decrease of 60.75% at 5000ΔT. In contrast, the opposite effect was observed for soft thresholding and DSR models, which diverged at a higher rate after L1-regularization was applied to their training procedure. In fact, DSR models reached a divergence percentage of 100% at only 2500ΔT.

In Fig. 9, a violin plot of the RFMSE of the non-diverging simulations for the different models is shown. The DSR model is not included because all of its predictions diverged, resulting in no available measurements under non-divergence

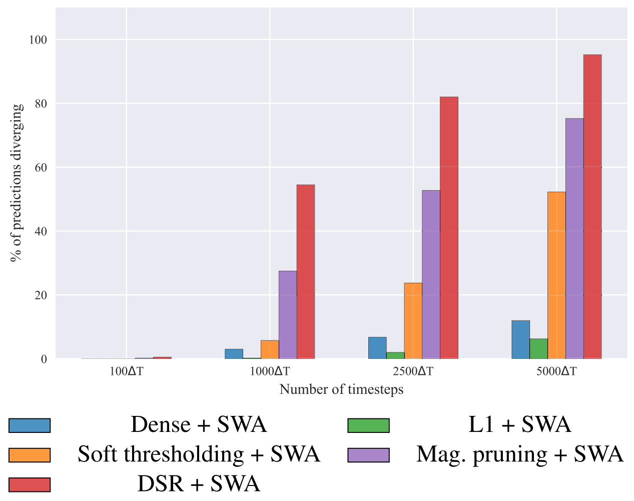


FIGURE 10. The figure shows a bar plot of divergence percentage for simulations using dense models combined with SWA and different sparsity methods. All models have a low divergence rate at early timesteps, with the rate decreasing towards the prediction horizon. The L1-regularized model has the lowest increase in divergence percentage.

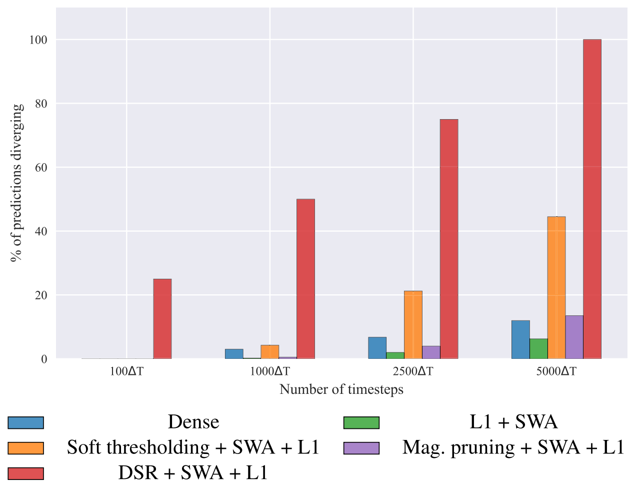


FIGURE 12. The figure shows a bar plot of divergence percentage for simulations by models applied some sparsity method together with L1-regularization and SWA. The simulations of all models have a low divergence rate at early stages, except for the simulations using the DSR models. The number of diverging simulations increases towards the simulation horizon, but the simulations by models with L1-regularization only combined with SWA and magnitude pruning with L1-regularization and SWA has a rather low divergence percentage through the whole simulation.

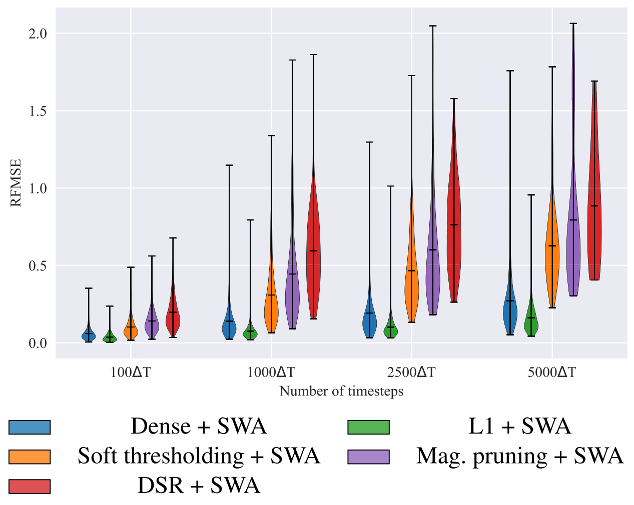


FIGURE 11. The figure shows a violin plot of the RFMSE of simulations performed by models combining sparsity methods and SWA. All models have a low error at the start with it increasing towards the end. The simulations of the models with L1-regularization and SWA maintain a low and dense error distribution throughout the whole simulation.

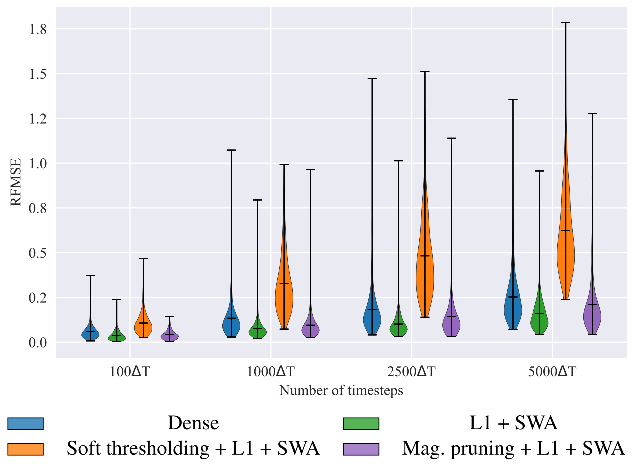


FIGURE 13. The figure shows a violin plot of RFMSE for simulations by models applied some sparsity method together with L1-regularization and SWA. The simulations by all models have a low error at early stages, with dense and low error distributions. The error increases through the simulations, but the simulations by models with L1-regularization only combined with SWA and magnitude pruning with L1-regularization and SWA has a rather low error and dense error distribution throughout the simulation horizon.

conditions. At $100\Delta T$, all models except for soft thresholding had approximately the same error distribution and mean, and interestingly, magnitude pruning had the lowest maximum error of all model types at this point. As the simulation progressed in time, all models exhibited increasing means, minimum values, and maximum values for their RFMSE, with the soft thresholding models increasing the most and L1-regularized models achieving the best performance in terms of mean and maximum RFMSE values, closely followed by L1 with magnitude pruning.

The application of soft thresholding combined with L1-regularization increased the sparsity in the final models from 56% to 71%, as shown in Table 4. The most interesting result from this set of experiments is the dramatic improvement in model performance when L1-regularization is combined with magnitude pruning. Since L1-regularization pulls weights towards zero and magnitude pruning removes a proportion of the smallest weights, it seems that the combination of these approaches results in models with a larger proportion of important and representative weights for the dynamics of the system. As L1 does not remove weights by itself the application of some mechanism, such as magnitude pruning, to effectively remove the irrelevant weights is necessary for the models to be actually sparse in terms of inference speed and computing cost.

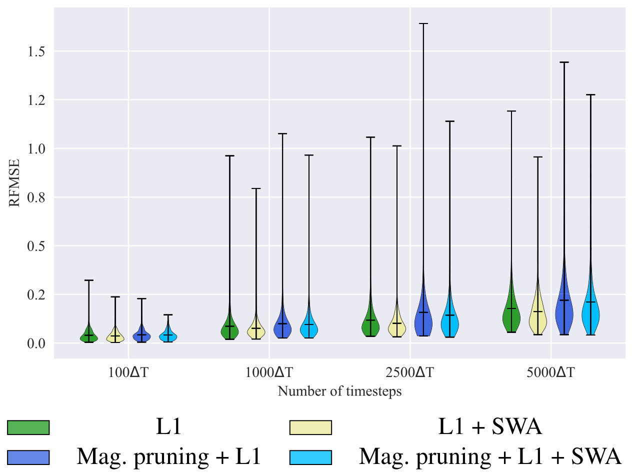


FIGURE 14. The figure shows the RFMSE of the dense and magnitude pruned models with L1-regularization only and with a combination of L1-regularization and SWA.

C. SWA

Fig. 10 illustrates the divergence of various methods when SWA is applied to their training processes. As shown in Table 2, the application of SWA to the training of dense, L1-regularized, and soft thresholding models appears to primarily affect their performance in the early stages of the simulation. From the table, it can be seen that using SWA during training yields an equal or better divergence percentage at $100\Delta T$ and $1000\Delta T$ for all models compared to the "vanilla" models. Notably, SWA results in a significant improvement in the performance of the DSR models at all ΔT values.

Fig. 11 further supports the results from the divergence plot and table. The plot shows all model types, including DSR. It can be seen that the RFMSE values are low for all models at early stages, but as time progresses the RFMSE of all models increases. The application of SWA seems to lower the densest part of the RFMSE distribution for the earlier timesteps.

It is interesting to note that SWA results in a performance increase for all models, particularly at early timesteps. Learning the dynamics of a complex and nonlinear system, such as the aluminum extraction process, is a challenging task. As such, it is likely that small changes in the weights of the networks, even at low learning rates, can result in significant changes in performance. In this case, the application of SWA seems to help the neural network find a better and more general solution that serves as a compromise between the different trajectories in the training data. However, since the error of the neural network is amplified as time progresses when integrating through the simulation, a more general solution may work well in the early stages of the simulation but may result in larger errors as time progresses.

D. SWA + L1

Fig. 12 shows the results of combining the different sparsity techniques with both L1 and SWA. As can be seen, combining L1 and SWA results in almost no diverging simulations at $100\Delta T$ for all model types except for DSR. Inspecting Table 2, it shows that 0% of soft thresholding models, magnitude pruning models, and dense, L1-regularized models are diverging. Combining L1 and SWA improves the performance at early timesteps for all models, compared to simply L1-regularizing the models.

Inspecting the violin plot of these models in Fig. 13, suggests a combination of the earlier discussed effects of L1 and SWA independently. Applying L1-regularization resulted in lower mean RFMSE values for most model types, and SWA resulted in more compact distributions at early stages than "vanilla" models. And, as it can be seen from Fig. 13, the distributions of the error at early stages seem to have its highest density centered lower for early timesteps. But as time progresses, especially soft thresholding has a significant increase in its RFMSE.

By combining SWA and L1-regularization the models are able to take advantage of the strengths of both methods. SWA helps the model's weights approach the global minimum, while L1-regularization helps the model learn the underlying

TABLE 2. The table contains the divergence percentage for the different model types.

Model type:	$100\Delta T$	$1000\Delta T$	$2500\Delta T$	$5000\Delta T$
Dense	0%	3%	6.75%	12%
Dense + SWA	0%	3%	6.75%	12%
L1	0%	0.75%	1.75%	6.25%
L1 + SWA	0%	0.25%	2.00%	6.25%
Soft thresholding	0%	6.75%	18.25%	38%
Soft thresholding + L1	0%	8.5%	27.5%	52%
Soft thresholding + SWA	0%	5.75%	23.75%	52.25%
Soft thresholding + L1 + SWA	0%	4.25%	21.25%	44.5%
Magnitude pruning	0.25%	30.5%	50%	71.75%
Magnitude pruning + L1	0%	0.75%	3.75%	11%
Magnitude pruning + SWA	0.25%	27.5%	52.75%	75.25%
Magnitude pruning + L1 + SWA	0%	0.5%	4%	13.5 %
DSR	3.75%	79.5%	94.5%	100%
DSR + L1	64.25%	99.25%	100%	100%
DSR + SWA	0.5%	54.5%	82%	95.25%
DSR + L1 + SWA	25%	50%	75%	100.00%

TABLE 3. The table contains the RFMSE of the different model types at the different timesteps. It is presented as the mean ± the standard deviation.

Model type:	$100\Delta T$	$1000\Delta T$	$2500\Delta T$	$5000\Delta T$
Dense	0.06 ± 0.04	0.13 ± 0.12	0.18 ± 0.14	0.25 ± 0.15
Dense + SWA	0.06 ± 0.04	0.14 ± 0.12	0.19 ± 0.18	0.27 ± 0.22
L1	0.04 ± 0.03	0.09 ± 0.08	0.12 ± 0.11	0.18 ± 0.13
L1 + SWA	0.04 ± 0.03	0.08 ± 0.06	0.10 ± 0.08	0.16 ± 0.10
Soft thresholding	0.10 ± 0.06	0.29 ± 0.19	0.42 ± 0.23	0.57 ± 0.25
Soft thresholding + L1	0.11 ± 0.06	0.31 ± 0.17	0.47 ± 0.22	0.63 ± 0.22
Soft thresholding + SWA	0.10 ± 0.07	0.31 ± 0.21	0.47 ± 0.24	0.63 ± 0.26
Soft thresholding + L1 + SWA	0.11 ± 0.06	0.33 ± 0.17	0.48 ± 0.24	0.63 ± 0.26
Magnitude pruning	0.17 ± 0.12	0.40 ± 0.20	0.55 ± 0.28	0.72 ± 0.31
Magnitude pruning + L1	0.04 ± 0.03	0.10 ± 0.10	0.16 ± 0.20	0.22 ± 0.19
Magnitude pruning + SWA	0.14 ± 0.08	0.44 ± 0.30	0.60 ± 0.37	0.79 ± 0.45
Magnitude pruning + L1 + SWA	0.04 ± 0.02	0.10 ± 0.08	0.14 ± 0.14	0.21 ± 0.16
DSR	0.25 ± 0.13	0.67 ± 0.26	1.01 ± 0.29	-
DSR + L1	0.50 ± 0.19	1.25 ± 0.44	-	-
DSR + SWA	0.20 ± 0.11	0.60 ± 0.27	0.76 ± 0.30	0.89 ± 0.38
DSR + L1 + SWA	0.53 ± 0.19	-	-	-

dynamics of the system, rather than just memorizing the data. In particular, magnitude pruning and dense L1-regularized models have been shown to benefit from the combination of L1-regularization and SWA. This is discussed further in section IV-E. However, the performance of soft thresholding and DSR models is not improved by the combination of L1-regularization and SWA, and they perform better when only the less detrimental technique is used.

E. L1-REGULARIZATION COMPARED TO L1-REGULARIZATION AND SWA FOR DENSE AND MAGNITUDE PRUNED MODELS

Combining L1 and SWA improves the performance of the dense and magnitude-pruned models, which can be seen in the comparison made in Fig. 13. It shows the RFMSE of the models with only L1-regularization applied during training and with both L1-regularization and SWA applied. From this plot, it is easier to see how the different models differ in performance.

It is evident from the plot that models with SWA applied exhibit a denser RFMSE distribution, particularly at $100\Delta T$. This indicates that SWA improves the model's generalization during training by averaging the weights of the best models. While the gap between regular L1 models and SWA models narrows over time, the accuracy of the L1 + SWA models remains consistently higher throughout the simulations.

TABLE 4. The table presents the sparsity of the different model types after training. It denotes the percentage of weights in the models that are 0.

Model type:	Mean sparsity:
Dense	0%
SWA	0%
L1	0%
L1 + SWA	0%
Soft thresholding	56%
Soft thresholding + L1	71%
Soft thresholding + SWA	56 %
Soft thresholding + L1 + SWA	71%
Magnitude pruning	33%
Magnitude pruning + L1	33%
Magnitude pruning + SWA	33%
Magnitude pruning + L1 + SWA	33%
DSR	50%
DSR + L1	50%
DSR + SWA	50%
DSR + L1 + SWA	50%

V. CONCLUSION AND FURTHER WORK

This study presents the application of sparsity-inducing techniques to neural networks to improve accuracy and reduce divergence when simulating complex dynamical systems. Techniques used include hard thresholding with magnitude pruning, soft thresholding with learnable sparsity, pruning and regrowing with dynamic sparse reparameterization, and L1 regularization. The impact of Stochastic Weight Averaging (SWA) was also evaluated. The methods were tested on a fully connected neural network and compared with dense L1-regularized networks using the aluminum extraction process as a test case. The main conclusions from this work are as follows.

- The application of L1 regularization to the training process was found to be the most crucial factor for model performance among the techniques explored in this study.
- The combination of magnitude pruning and L1 regularization resulted in slightly inferior performance compared to L1 alone, suggesting that L1 regularization strengthens critical connections in the network, while magnitude pruning removes redundant ones.
- Soft threshold reparameterization for learnable sparsity and dynamic sparse reparameterization, techniques

designed to reduce the number of parameters in large image classification models, appear to be ill-suited for smaller and simpler models. In particular, dynamic sparse reparameterization appears to remove important connections during training without providing adequate means for regrowing others to compensate.
- The application of SWA during training increases performance for both the L1-regularized dense model and the magnitude pruned model. It also reduces the number of diverging simulations and RFMSE for all models in an early to medium time horizon.

The results of this study indicate that L1-regularized models perform best, supporting the notion that imposing sparsity on neural networks can improve their performance by forcing them to learn only the most important connections. However, it is clear that the choice of sparsity technique is not arbitrary, and the experiments in this work suggest that simpler methods are typically more effective. There are many more sparsity-imposing techniques available that have yet to be applied in the context of dynamical systems. Evaluating their performance remains one of the future tasks.

Reproducibility: The code implemented for this work is publicly available in https://www.github.com/emilhaugstvedt/sparsity. This includes all model training files and all trained models.

REFERENCES

[1] Ž. Avsec, V. Agarwal, D. Visentin, J. R. Ledsam, A. Grabska-Barwinska, K. R. Taylor, Y. Assael, J. Jumper, P. Kohli, and D. R. Kelley, "Effective gene expression prediction from sequence by integrating long-range interactions," *Nature Methods*, vol. 18, no. 10, pp. 1196–1203, Oct. 2021. [Online]. Available: https://www.biorxiv.org/content/early/2021/04/08/2021.04.07.438649

[2] S. Li, W. Song, L. Fang, Y. Chen, P. Ghamisi, and J. A. Benediktsson, "Deep learning for hyperspectral image classification: An overview," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 9, pp. 6690–6709, Sep. 2019.

[3] X. Yang, S. Zhang, J. Liu, Q. Gao, S. Dong, and C. Zhou, "Deep learning for smart fish farming: Applications, opportunities and challenges," *Rev. Aquaculture*, vol. 13, no. 1, pp. 66–90, Jan. 2021, doi: 10.1111/raq.12464.

[4] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang, and Y. Miao, "Review of image classification algorithms based on convolutional neural networks," *Remote Sens.*, vol. 13, no. 22, p. 4712, Nov. 2021. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85119909628&doi=10.3390%2frs13224712&partnerID=40&md5=e27668801179624a5636a7b3ed3e892a

[5] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 7, pp. 3523–3542, Jul. 2022. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85100948197&doi=10.1109%2fTPAMI.2021.3059968&partnerID=40&md5=115a97279a77fb75cf00d20ca1578417

[6] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, "A survey of modern deep learning based object detection models," *Digit. Signal Process.*, vol. 126, Jun. 2022, Art. no. 103514. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85126330193&doi=10.1016%2fj.dsp.2022.103514&partnerID=40&md5=62a6fc1c0c7a3dc38df23007b657d801

[7] S. W. Zamir, A. Arora, S. Khan, M. Hayat, F. S. Khan, and M. Yang, "Restormer: Efficient transformer for high-resolution image restoration," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 5718–5729. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85137771434&doi=10.1109%2fCVPR52688.2022.00564&partnerID=40&md5=47efb85e7c5415de0fc58160891e7bde

[8] OpenAi. *Dall-E 2*. Accessed: Dec. 13, 2022. [Online]. Available: https://openai.com/dall-e-2/

[9] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with CLIP latents," 2022, *arXiv:2204.06125*.

[10] OpenAi. *ChatGPT: Optimizing Language Models for Dialogue*. Accessed: Dec. 13, 2022. [Online]. Available: https://openai.com/blog/chatgpt/

[11] K. S. Narendra and K. Parthasarathy, "Neural networks and dynamical systems," *Int. J. Approx. Reasoning*, vol. 6, no. 2, pp. 109–131, Feb. 1992. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0888613X9290014Q

[12] C. Møldrup Legaard, T. Schranz, G. Schweiger, B. Falay, C. Gomes, A. Iosifidis, M. Abkar, and P. Gorm Larsen, "Constructing neural network-based models for simulating dynamical systems," 2021, *arXiv:2111.01495*.

[13] X. Ying, "An overview of overfitting and its solutions," *J. Phys., Conf. Ser.*, vol. 1168, Feb. 2019, Art. no. 022022, doi: 10.1088/1742-6596/1168/2/022022.

[14] Y. Chauvin, "A back-propagation algorithm with optimal use of hidden units," in *Advances in Neural Information Processing Systems*, vol. 1, D. Touretzky, Ed. Burlington, MA, USA: Morgan-Kaufmann, 1988. [Online]. Available: https://proceedings.neurips.cc/paper/1988/file/9fc3d7152ba9336a670e36d0ed79bc43-Paper.pdf

[15] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems*, vol. 5, S. Hanson, J. Cowan, and C. Giles, Eds. Burlington, MA, USA: Morgan-Kaufmann, 1992. [Online]. Available: https://proceedings.neurips.cc/paper/1992/file/303ed4c69846ab36c2904d3ba8573050-Paper.pdf

[16] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, vol. 2, D. Touretzky, Ed. Burlington, MA, USA: Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf

[17] A. P. Trischler and G. M. T. D'Eleuterio, "Synthesis of recurrent neural networks for dynamical system simulation," *Neural Netw.*, vol. 80, pp. 67–78, Aug. 2016.

[18] M. V. Egorchev and Y. V. Tiumentsev, "Semi-empirical neural network based approach to modelling and simulation of controlled dynamical systems," *Proc. Comput. Sci.*, vol. 123, pp. 134–139, Jan. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918300231

[19] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015, *arXiv:1506.02626*.

[20] A. Kusupati, V. Ramanujan, R. Somani, M. Wortsman, P. Jain, S. Kakade, and A. Farhadi, "Soft threshold weight reparameterization for learnable sparsity," 2020, *arXiv:2002.03231*.

[21] H. Mostafa and X. Wang, "Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization," 2019, *arXiv:1902.05967*.

[22] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," 2021, *arXiv:2102.00554*.

[23] M. Nagahara, *Sparsity Methods for Systems and Control*, Sep. 2020.

[24] K. Kaheman, J. N. Kutz, and S. L. Brunton, "SINDy-PI: A robust algorithm for parallel implicit sparse identification of nonlinear dynamics," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 476, no. 2242, Oct. 2020, Art. no. 20200279.

[25] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Sparse identification of nonlinear dynamics for model predictive control in the low-data limit," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 474, no. 2219, Nov. 2018, Art. no. 20180335.

[26] S. Pan and K. Duraisamy, "Data-driven discovery of closure models," *SIAM J. Appl. Dyn. Syst.*, vol. 17, no. 4, pp. 2381–2413, Jan. 2018.

[27] C. Gallicchio and A. Micheli, "Fast and deep graph neural networks," in *Proc. AAAI Conf. Artif. Intell.*, Apr. 2020, vol. 34, no. 4, pp. 3898–3905.

[28] M. Kafashan, A. Nandi, and S. Ching, "Relating observability and compressed sensing of time-varying signals in recurrent linear networks," *Neural Netw.*, vol. 83, pp. 11–20, Nov. 2016.

[29] S. Strogatz, *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Boca Raton, FL, USA: CRC Press, 2018. [Online]. Available: https://books.google.no/books?id=1kpnDwAAQBAJ

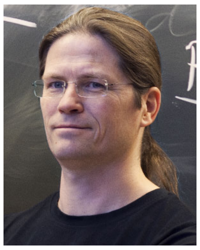
[30] E. T. B. Lundby, A. Rasheed, J. T. Gravdahl, and I. J. Halvorsen, "Sparse deep neural networks for modeling aluminum electrolysis dynamics," *Appl. Soft Comput.*, vol. 134, Feb. 2023, Art. no. 109989.
[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org
[32] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.
[33] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[34] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *Towards Data Sci.*, vol. 6, no. 12, pp. 310–316, 2017.
[35] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," 2016, *arXiv:1611.03530*.
[36] D. L. Donoho, "De-noising by soft-thresholding," *IEEE Trans. Inf. Theory*, vol. 41, no. 3, pp. 613–627, May 1995.
[37] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, "Averaging weights leads to wider optima and better generalization," 2018, *arXiv:1803.05407*.
[38] T. Garipov, P. Izmailov, D. Podoprikhin, D. Vetrov, and A. G. Wilson, "Loss surfaces, mode connectivity, and fast ensembling of DNNs," 2018, *arXiv:1802.10026*.
[39] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, pp. 686–707, Feb. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999118307125
[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, *arXiv:1512.03385*.
[41] OpenAi and A. Pilipiszyn. *GPT-3—Powers the Next Generation of Apps*. Accessed: Dec. 10, 2022. [Online]. Available: https://openai.com/blog/gpt-3-apps/
[42] H. Binous and N. Zakia. *Duffing Oscillator*. [Online]. Available: http://demonstrations.wolfram.com/DuffingOscillator/
[43] R. Cangelosi. *Lotka-Volterra Competition Model*. [Online]. Available: http://demonstrations.wolfram.com/LotkaVolterraCompetitionModel/
[44] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *IEEE Trans. Nucl. Sci.*, vol. 44, no. 3, pp. 1464–1468, Jun. 1997.
[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent.*, Y. Bengio and Y. LeCun, Eds. San Diego, CA, USA, May 2015.
[46] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," 2015, *arXiv:1502.01852*.
[47] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*.

EMIL JOHANNESEN HAUGSTVEDT is currently pursuing the master's degree with the Department of Engineering Cybernetics, Norwegian University of Science and Technology (NTNU). He is working on the intersection between machine learning and physical systems. His main research interests include how machine learning can be used to increase the interpretability, understanding, and performance of dynamical system simulations.

ALBERTO MIÑO CALERO received the bachelor's degree in computer science from the University of Córdoba and the master's degree in computer science and technology from the Charles III University of Madrid. He is currently pursuing the Ph.D. degree with the Department of Engineering Cybernetics, Norwegian University of Science and Technology. He is working on explainable AI applied to neural networks, and how to provide explainability layers to black-box models to gain a better understanding of their behavior, mainly in the context of dynamical systems.

ERLEND TORJE BERG LUNDBY (Member, IEEE) received the master's degree in marine technology and the Ph.D. degree in engineering cybernetics from NTNU. He is currently working on machine learning and system identification in closed-loop systems. An important part of his research interest includes reducing the amount of data required to achieve acceptable model accuracy. He has focused on model complexity, model structure, hybrid modeling, and active learning to address this issue.

ADIL RASHEED is currently a Professor with the Department of Engineering Cybernetics, Norwegian University of Science and Technology. There, he works to advance the development of novel hybrid methods that combine big data, physics-driven modeling, and data-driven modeling in the context of real-time automation and control. In addition, he holds a part-time senior scientist position with the Department of Mathematics and Cybernetics, SINTEF Digital, where he was the Leader of the Computational Sciences and Engineering Group, from 2012 to 2018. He is leading the digital twin and asset management-related work in the FME NorthWind Center. His contributions in these roles have been the development and advancement of both hybrid analysis and modeling and big data cybernetics concepts. Over the course of his career, he has been the driving force behind numerous projects focused on different aspects of digital twin technology, ranging from autonomous ships to wind energy, aquaculture, drones, business processes, and indoor farming.


JAN TOMMY GRAVDAHL (Senior Member, IEEE) received the Siv.ing. and Dr.-Ing. degrees in engineering cybernetics from the Norwegian University of Science and Technology (NTNU), Trondheim, Norway, in 1994 and 1998, respectively. Since 2005, he has been a Professor with the Department of Engineering Cybernetics, NTNU, where he was the Head of the Department, from 2008 to 2009. He has supervised the graduation of 160 M.Sc. and 15 Ph.D. candidates. He has published five books and more than 300 papers in international conferences and journals. His current research interests include mathematical modeling and nonlinear control in general, in particular, applied to turbomachinery, marine vehicles, spacecraft, robots, and high-precision mechatronic systems. In 2000 and 2017, he received the IEEE Transactions on Control Systems Technology Outstanding Paper Award. He has been on the editorial board and IPC for numerous international conferences. He served as an Associate Editor for *Simulation Modelling Practice and Theory*, from 2007 to 2011. He served as a Senior Editor for *IFAC* and *Mechatronics*, from 2016 to 2020. Since 2020, he has been an Associate Editor of IEEE Transactions on Control Systems Technology.