Hybrid Physics-informed Neural Networks for Dynamical Systems

Renato Giorgiani do Nascimento
University of Central Florida

HYBRID PHYSICS-INFORMED NEURAL NETWORKS FOR DYNAMICAL SYSTEMS

by

RENATO G. NASCIMENTO
B.S. Universidade Estadual Paulista, 2015

A thesis submitted in partial fulfilment of the requirements
for the degree of Master of Science in Aerospace Engineering
in the Department of Mechanical and Aerospace Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2020

ABSTRACT

Ordinary differential equations can describe many dynamic systems. When physics is well understood, the time-dependent responses are easily obtained numerically. The particular numerical method used for integration depends on the application. Unfortunately, when physics is not fully understood, the discrepancies between predictions and observed responses can be large and unacceptable. In this thesis, we show how to directly implement integration of ordinary differential equations through recurrent neural networks using Python. We leveraged modern machine learning frameworks, such as TensorFlow and Keras. Besides offering basic models capabilities (such as multilayer perceptrons and recurrent neural networks) and optimization methods, these frameworks offer powerful automatic differentiation. With that, our approach's main advantage is that one can implement hybrid models combining physics-informed and data-driven kernels, where data-driven kernels are used to reduce the gap between predictions and observations. In order to illustrate our approach, we used two case studies. The first one consisted of performing fatigue crack growth integration through Euler's forward method using a hybrid model combining a data-driven stress intensity range model with a physics-based crack length increment model. The second case study consisted of performing model parameter identification of a dynamic two-degree-of-freedom system through Runge-Kutta integration. Additionally, we performed a numerical experiment for fleet prognosis with hybrid models. The problem consists of predicting fatigue crack length for a fleet of aircraft. The hybrid models are trained using full input observations (far-field loads) and very limited output observations (crack length data for only a portion of the fleet). The results demonstrate that our proposed physics-informed recurrent neural network can model fatigue crack growth even when the observed distribution of crack length does not match the fleet distribution.

To my wife Larissa and my daughter Sara.

ACKNOWLEDGMENTS

I want to express my gratitude to my parents and siblings. The good education they gave me, their love, support, and the incentive was and will always be fundamental in my life. I am also immensely thankful to my wife Larissa for dreaming my dreams with me and for the love she has devoted to me all these years.

I am thankful to my academic advisor Dr. Felipe Viana for his guidance, patience, and encouragement. I would also like to thank my advisory committee members, Dr. Jihua Gou and Dr. Helen Huang. I am grateful for their willingness to serve on my committee and for the constructive criticism they have provided.

I wish to thank all my colleagues for their friendship and support. Thanks, Andre, Arinan, Kajetan, and Yigit.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Motivation

Predictive models [1, 2, 3, 4] are often used to model cumulative distress in critical components (diagnosis and prognosis) of engineering assets (e.g., aircraft, jet engines, and wind turbines). These models usually leverage data coming from design, manufacturing, configuration, online sensors, historical records, inspection, maintenance, location, and satellite data). In terms of modeling, we believe most practitioners would agree that (a) machine learning models offer flexibility but tend to require large amounts of data, and (b) physics-based models are grounded on first-principles and require good understanding of physics of failure and degradation mechanisms. In practice, the decision between machine learning and physics-based models depends on factors such as existing knowledge (maybe even legacy models), amount and nature of data, accuracy and computational requirements, timelines for implementation, etc. The interested reader can find a discussion on how these concepts apply to defense systems in [5, 6] and examples of industrial and commercial applications in [7, 8, 9, 10].

Deep learning and physics-informed neural networks [11, 12, 13, 14] have received growing attention in science and engineering over the past few years. The fundamental idea, particularly with physics-informed neural networks, is to leverage laws of physics in the form of differential equations in the training of neural networks. This is fundamentally different than using neural networks as surrogate models trained with data collected at a combination of inputs and output values. Physics-informed neural networks can be used to solve the forward problem (estimation of response) and/or the inverse problem (model parameter identification).

Although there is no consensus on nomenclature or formulation, we see two different and very broad approaches to physics-informed neural network. There are those using neural network as approximate solutions for the differential equations [13, 15, 16, 17]. Essentially, through collocation points, the neural network hyperparameters are optimized to satisfy initial/boundary conditions as well as the constitutive differential equation itself. For example, Raissi et al. [15] present an approach for solving and discovering the form of differential equations using neural networks, which has a companion GitHub repository (`https://github.com/maziarraissi/PINNs`) with detailed and documented Python implementation. Authors proposed using deep neural networks to handle the direct problem of solving differential equations through the loss function (functional used in the optimization of hyperparameters). The formulation is such that neural networks are parametric trial solutions of the differential equation and the loss function accounts for errors with respect to initial/boundary conditions and collocation points. Authors also present a formulation for learning the coefficients of differential equations given observed data (i.e., calibration). The proposed method is applied to both, the Schroedinger equation, a partial differential equation utilized in quantum mechanics systems, and the Allen-Cahn equation, an established equation for describing reaction-diffusion systems. Pan and Duraisamy [17] introduced a physics informed machine learning approach to learn the continuous-time Koopman operator. Authors apply the derived method to nonlinear dynamical systems, in particular within the field of fluid dynamics, such as modeling the unstable wake flow behind a cylinder. In order to derive the method, authors used a measure-theoretic approach to create a deep neural network. Both differential and recurrent model types are derived, where the latter is used when discrete trajectory data can be obtained whereas the differential form is suitable when governing equations are disposable. This physics-informed neural network approach shows its strength regarding uncertainty quantification and is robust against noisy input signal.

Alternatively, there are those building hybrid models that directly code reduced order physics-informed models within deep neural networks [18, 19, 20, 21, 22]. This implies that the computational cost of these physics-informed kernels have to be comparable to the linear algebra found in neural network architectures. It also means that tuning of the physics-informed kernel hyperparameters through backpropagation requires that adjoints to be readily available (through automatic differentiation [23], for example). For example, Yucesan and Viana [19] proposed a hybrid modeling approach which combines reduced-order models and machine learning for improving the accuracy of cumulative damage models used to predict wind turbine main bearing fatigue. The reduce-order models capture the behavior of bearing loads and bearing fatigue; while machine learning models account for uncertainty in grease degradation. The model was successfully used to predict grease degradation and bearing fatigue across a wind park; and with that, optimize the regreasing intervals. Karpatne et al. [21] presented an interesting taxonomy for what authors called theory-guided data science. In the paper, they discuss how one could augment machine learning models with physics-based domain knowledge and walk from simple correlation-based models, to hybrid models, to fully physics-informed machine learning (such as in solving differential equations directly). Authors discuss examples in hydrological modeling, computational chemistry, mapping surface water dynamics, and turbulence modeling.

In this thesis, we focus on discussing a Python implementation for hybrid physics-informed neural networks. We believe these hybrid implementations can have an impact in real-life applications, where reduced order models capturing the physics are available and well adopted. Most of the time, the computational efficiency of reduced order models comes at the cost of loss of physical fidelity. Hybrid implementations of physics-informed neural networks can help reducing the gap between predictions and observed data.

Our approach starts with the analytical formulation and passes through the numerical integration method before landing in the neural network implementation. Depending on the application, different numerical integration methods can be used. While this is an interesting topic, it is not the focus of this work. Instead, we focus on how to move from analytical formulation to numerical implementation of the physics-informed neural network model.

We address the implementation of ordinary differential equation solvers using two case studies in engineering. Fatigue crack propagation is used as an example of first order ordinary differential equations. In this example, we show how physics-informed neural networks can be used to mitigate epistemic (model-form) uncertainty in reduced order models. Forced vibration of a 2 degree-of-freedom system is used as an example of a system of second order ordinary differential equations. In this example, we show how physics-informed neural networks can be used to estimate model parameters of a physical system.

Literature Review

The literature on the use of traditional and modern machine learning methods for diagnosis and prognosis is rich. For example, Si et al. [24] reviewed statistical data driven approaches for prognosis that rely only on available past observed data and statistical models (regression, Brownian motion with drift, gamma processes, Markovian-based models, stochastic filtering-based models, hazard models, and hidden Markov models). Tamilselvan and Wang [25] discussed a multi-sensor health diagnosis and prognosis method using deep belief networks. They demonstrate how deep belief networks can model the probabilistic transition between health state and damaged state in aircraft engines and power transformers. Interestingly, Son et al. [26] reported how they solved the same aircraft engine problem using Wiener process combined with principal component analysis. Susto et al. [27] discussed how to approach remaining useful life estimation using ensembles of classifiers. They based their work on traditional support vector machines and k-nearest neighbors and tested it on estimating remaining useful life of tungsten filaments used in ion implantation (important in semiconductor fabrication). Khan and Yairi [28] reviewed the application of deep learning in structural health management (simple autoencoders, denoising autoencoder, variational autoencoders, deep belief networks, restricted and deep Boltzmann machines, convolutional neural networks, and purely data-driven versions of recurrent neural networks, including the long short-term memory and gated recurrent units). They found that most approaches are still application specific (unfortunately, they did not find a clear way to select, design, or implement a deep learning architecture for structural health management). They also advise that a trade-off study should be performed when considering complexity and computational cost. Stadelmann et al. [29] brought an interesting discussion on deep learning applied to industry. Among the case studies, they discussed predictive maintenance with machine learning approaches such as support vector machines, Gaussian mixtures, principal component analysis and how they compare with deep learning approaches such fully connected and variational as autoencoders.

There is also considerable research on the use of physics-based methods for diagnosis and prognosis. Although literature tends to be very application specific, most authors use a physics-based model for damage accumulation and a statistical/machine learning technique for parameter estimation and uncertainty quantification. For example, Daigle and Goebel [30] formulated physics-based prognostics as joint state-parameter estimation problem, in which the state of a system along with parameters describing the damage progression are estimated. This is followed by a prediction problem, in which the joint state-parameter estimate is propagated forward in time to predict end of life and remaining useful life. They demonstrate their methodology in the estimation of remaining useful life of centrifugal pump used for liquid oxygen loading located at the Kennedy Space Center. Li et al. [31] modeled fatigue crack growth on the leading edge of an aircraft wing. The model was updated through dynamic Bayesian networks with observed data (including both damage and loads). The updated model was used in diagnosis and prognosis of an aircraft digital twin (virtual representation of the physical aircraft). Ling et al. [32] also modeled fatigue crack growth on the leading edge of an aircraft wing. However, instead of only estimating remaining useful life, they used information gain theory to evaluate the usefulness of aircraft component inspection.

This helps deciding whether or not inspection is worthwhile (e.g., model improvement justifying the cost). A dynamic Bayesian network tracks and forecasts fatigue crack growth and the detection of a crack is modeled through probability of detection. Information gain per cost of inspection is used to identify the optimal timing of next inspection. Yucesan and Viana [33] modeled main bearing fatigue in onshore wind turbines, coupling damange models for bearing raceway and grease (lubricant). Their results demonstrated that, although bearing fatigue is a secondary life limiting factor, it can still contribute significantly to bearing failures. They also showed how to use their physics-based cumulative damage model to promote component life extension. Berri et al. [34] proposed a framework for prognosis including signal acquisition, fault detection and isolation, and remaining useful life estimation. In order to keep computational cost manageable, they proposed using strategies for signal processing combined with physical models of different fidelity and machine learning techniques. They successfully tested their approach on aircraft electromechanical actuator for secondary flight controls. Byington et al. [35] presented a study on the use of neural networks for prognosis of aircraft actuator components. The framework covered tasks such as feature extraction, data cleaning, classification, information fusion, and prognosis. They successfully demonstrated their approach on F-18 stabilator electro-hydraulic servo valves. Jacazio and Sorli [36] presented an enhanced particle filter framework for prognosis of electro-mechanical flight controls actuators. They achieved promising results and showed the benefits of their approach as compared to other published methods.

Using artificial neural networks for solving differential equations with applications in engineering is a relatively old concept [37, 38, 39, 40]. Nevertheless, the unparalleled computational power available these days contributed to the popularization of machine learning in engineering applications [41, 42, 43]. The scientific community has been studying and proposing deep learning architectures that leverage mathematical models based in physics and engineering principles [44, 16, 13, 45, 46]. Differential equations are used to train multi-layer perceptrons and recurrent neural networks. The idea is to use the physics laws (in the form of differential equations) to help handling the reduced number of data points, and constrain the hyper-parameter space. With incompressible fluids, for example, this is done by discarding non-realistic solutions violating the conservation of mass principle. Raissi [47] approximated the unknown of solution of partial differential equations by two deep neural networks. The first network acts as a prior on the unknown solution (enabling to avoid ill-conditioned and unstable numerical differentiations). The second network works as a fine approximation to the spatiotemporal solution. The methodology was tested on a variety of equations used in fluid mechanics, nonlinear acoustics, gas dynamics, and other fields. Wu et al. [48] discussed, in depth, how to augment turbulence models with physics-informed machine learning. They demonstrated a procedure for computing mean flow features based on the integrity basis for mean flow tensors and propose using machine learning to predict the linear and nonlinear parts of the Reynolds stress tensor separately. They used the flow in a square duct and the flow over periodic hills to evaluate the performance of the proposed method. Hesthaven and Ubbiali [49] proposed a non-intrusive reduced-basis method (using proper orthogonal decomposition and neural networks) for parametrized steady-state partial differential equations. The method extracts a reduced basis from a collection of snapshots through proper orthogonal decomposition and employs multi-layer perceptrons to approximate the coefficients of the reduced model. They successfully tested the proposed method on the nonlinear Poisson equation in one and two spatial dimensions, and on two-dimensional cavity viscous flows, modeled through the steady incompressible Navier–Stokes equations. Swischuk et al. [50] demonstrated through case studies (predictions of the flow around an airfoil and structural response of a composite panel) that proper orthogonal decomposition is an effective way to parametrize a high-dimensional output quantity of interest in order to define a low-dimensional map suitable for data-driven learning. They tested a variety of machine learning methods such as artificial neural networks, multivariate polynomial regression, k-nearest neighbor, and decision trees. The interested reader can also find literature on Gaussian processes [51, 45].

As we just discussed, there are several ways to build physics-informed machine learning models. In this work, we focus on neural network models suitable for solving ordinary differential equations (describing time-dependent quantities of interest). Chen et al. [13] demonstrated that deep neural networks can approximate dynamical systems which response comes out of integrating ordinary differential equations. In their approach deep neural networks represent very fine discretization along the lines of a very fine Euler integration. This is particularly applicable to recurrent neural networks and residual networks. Interestingly, the loss function operates on top of the ordinary differential equation solver. Therefore, training data is generated through adjoint methods (automatic differentiation). Kani and Elsheikh [52, 46] introduced the deep residual recurrent neural networks where a fixed number of layers are stacked together to minimize the residual (or reduced residual) of the physical model under consideration. To reduce the computational complexity associated with high-fidelity numerical simulations of physical systems (which generate the training data), they also used proper orthogonal decomposition. They demonstrated their approach with the simulation of a two-phase (oil and water) flow through porous media over a two-dimensional domain.

Scope and Organization of this Research

The research, conducted in the context of a Master thesis, has the following main objectives:

- Develop a framework for hybrid modeling of systems described by ordinary differential equations. In these hybrid models, the physics is given by the equations that govern the system dynamics and data-driven kernels are used to compensate for model simplifications.
- Implement and demonstrate the proposed framework in relevant engineering examples, including systems described by first and second order ordinary differential equations.

- Illustrate a comparison between the proposed framework against state-of-the art deep learning approaches.

The organization of this work is as follows. Chapter 2 gives an overview of neural networks, more specifically, how its nodes implement complex operations in a graph. It includes a brief description of how neural networks are trained and introduces recurrent neural networks concepts and known designs.

Chapter 3 discusses the implementation of the hybrid physics-informed neural network for ordinary differential equations. It consolidates the implementation with Python code fragments and case studies for both first and second order differential equations.

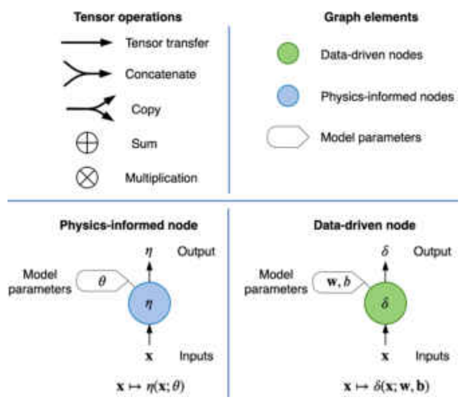
Chapter 4 offers an in-depth analysis of a numerical experiment for fleet prognosis with the hybrid model and compares it with state-of-the-art pure data-driven methods.

Finally, Chapter 5 highlights the present research work's major conclusions and portrays the future work scope on this topic.

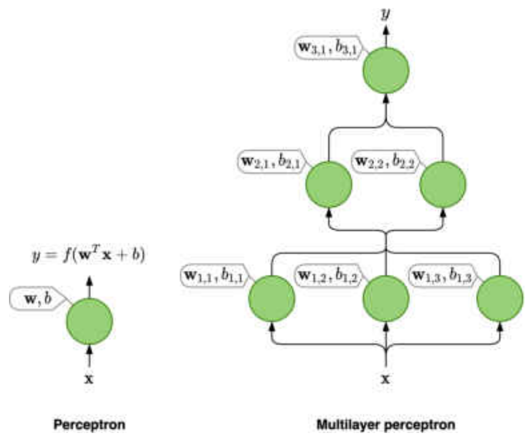
CHAPTER 2: BACKGROUND ON MACHINE LEARNING

Neural networks as directed graph models

Figure 2.1a introduces the notation and elements we used in this work. Tensors are used to represent inputs and outputs. The basic tensor operators include tensor transfer (which takes the tensor from one node of the graph to another), concatenation, copy, as well as algebraic operators, such as sum and multiplication. Nodes implement complex operations taking tensors as inputs and returning tensors as outputs. As we will describe later, nodes can implement physics-informed models. Traditionally, in neural networks, nodes implement data-driven models. As shown in Fig. 2.1b, a directed graph consists of finitely many nodes and edges, with edges directed from one node to another [53, 54]. We can use the idea to represent popular neural network architectures such as the perceptron or multilayer perceptron.



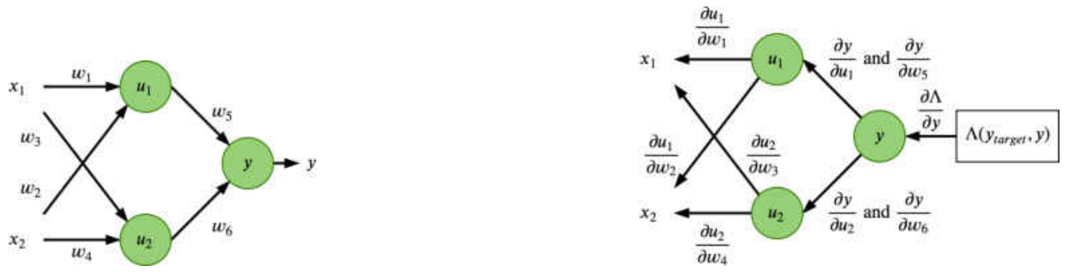
(a) Notation.


$y = f(\mathbf{w}^T\mathbf{x} + b)$
$\mathbf{w}, b$
$\mathbf{x}$
Perceptron
y
$\mathbf{w}_{3,1}, b_{3,1}$
$\mathbf{w}_{2,1}, b_{2,1}$
$\mathbf{w}_{2,2}, b_{2,2}$
$\mathbf{w}_{1,1}, b_{1,1}$
$\mathbf{w}_{1,2}, b_{1,2}$
$\mathbf{w}_{1,3}, b_{1,3}$
$\mathbf{x}$
Multilayer perceptron

(b) Perceptron and multilayer perceptron, $f(.)$ is an activation function.

Figure 2.1: Graph representation of neural networks.

As we will detail in Chapter 3, we propose directly implementing ordinary differential equations


x_1
w_1
u_1
w_3
w_5
y
w_2
x_2
w_4
u_2
w_6
$\frac{\partial u_1}{\partial w_1}$
$\frac{\partial y}{\partial u_1}$ and $\frac{\partial y}{\partial w_5}$
$\frac{\partial \Lambda}{\partial y}$
$\Lambda(y_{target}, y)$
$\frac{\partial u_1}{\partial w_2}$
$\frac{\partial u_2}{\partial w_3}$
$\frac{\partial y}{\partial u_2}$ and $\frac{\partial y}{\partial w_6}$
$\frac{\partial u_2}{\partial w_4}$

(a) Prediction (forward pass). (b) Training (backward pass).

Figure 2.2: Backpropagation overview. In prediction, inputs are fed forward, generating the activations of hidden layers u_i and output layer y. In training, partial derivatives are fed backward, generating the gradient of the loss function with respect to the weights, $\nabla\Lambda = \left[\frac{\partial\Lambda}{\partial w_1} \dots \frac{\partial\Lambda}{\partial w_6}\right]^T$.

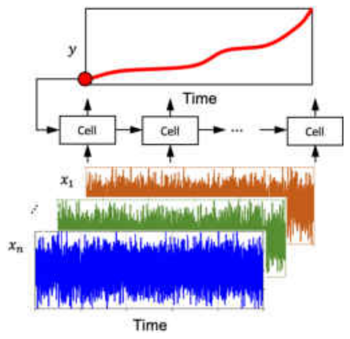
that describe the physics of a problem as deep neural networks using directed graph models. Therefore, optimization of hyperparameters is done through backpropagation. As illustrated in Fig. 2.2, there are essentially two steps in every iteration of the optimization algorithm. First, training data is fed forward, generating the corresponding outputs (Fig. 2.2a), prediction error, and finally the loss function. Then, the loss function adjoint is propagated backward (through the chain rule) giving the gradient with respect to the parameters to be optimized. Figure 2.2b is a graphical representation of backward pass for the multilayer perceptron. Even though the figure only shows the weight hyper-parameters ($\mathbf{w}$), the formulation can be extended for the case where each perceptron also has a bias term (b). Formally, from Fig. 2.2b, the gradient of the loss function with respect to the weights can be written as

$$\begin{aligned} \frac{\partial\Lambda}{\partial w_1} &= \frac{\partial\Lambda}{\partial y}\frac{\partial y}{\partial u_1}\frac{\partial u_1}{\partial w_1}, & \frac{\partial\Lambda}{\partial w_2} &= \frac{\partial\Lambda}{\partial y}\frac{\partial y}{\partial u_1}\frac{\partial u_1}{\partial w_2}, & \frac{\partial\Lambda}{\partial w_3} &= \frac{\partial\Lambda}{\partial y}\frac{\partial y}{\partial u_2}\frac{\partial u_2}{\partial w_3}, \\ \frac{\partial\Lambda}{\partial w_4} &= \frac{\partial\Lambda}{\partial y}\frac{\partial y}{\partial u_2}\frac{\partial u_2}{\partial w_4}, & \frac{\partial\Lambda}{\partial w_5} &= \frac{\partial\Lambda}{\partial y}\frac{\partial y}{\partial w_5}, \text{ and} & \frac{\partial\Lambda}{\partial w_6} &= \frac{\partial\Lambda}{\partial y}\frac{\partial y}{\partial w_6}. \end{aligned} \tag{2.1}$$

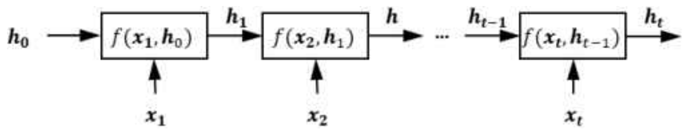
Figure 2.2 also gives insight into two important aspects of our framework. First, the forward pass has to be implemented with the available tensor operations. Second, in the backward pass, it is important to have the adjoints readily available. With that in place, one can start implementing the ordinary differential equations as deep neural networks.

Recurrent neural networks

Recurrent neural networks are specially suitable for dynamical systems [55, 56, 57]. They extend traditional feed forward networks to handle time-dependent responses, as shown in Fig. 2.3a. Recurrent neural networks have been used to model time-series [58], speech recognition [59], diagnosis and prognosis [60, 61, 62, 63], and many other applications.



(a) Basic idea.



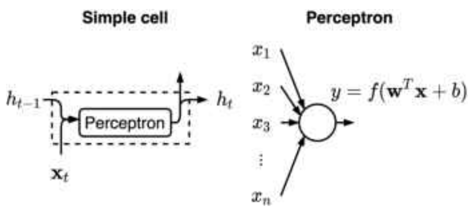
(b) Recurrent neural network (RNN). The function $f(\mathbf{h}_{t-1}, \mathbf{x}_t)$, a.k.a. the RNN cell, implements the transition from step to step throughout the time series.

Figure 2.3: Recurrent neural network. Cells repeatedly apply transformation to outputs, y_t, and inputs, $\mathbf{x}_t$.

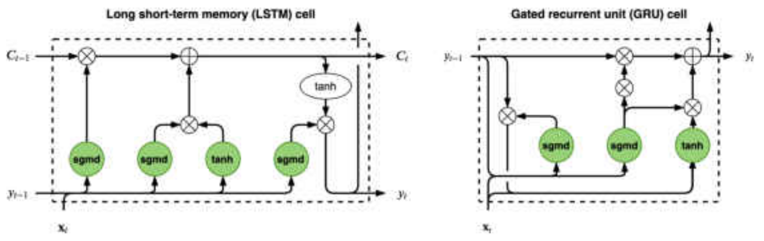
As illustrated in Fig. 2.3b, in every time step t, recurrent neural networks apply a transformation to a state $\mathbf{h}$ in the following fashion:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t), \tag{2.2}$$

where $t \in [0, \ldots, T]$ represent the time discretization, $\mathbf{h} \in \mathbb{R}^{n_h}$ are the states representing the



(a) Simple cell and perceptron



(b) LSTM and GRU cells.

Figure 2.4: Detailed recurrent neural networks cells. In the Long short-term memory (LSTM) and gated recurrent unit (GRU) cells, green “sgmd” and “tanh” circles are perceptrons with sigmoid and tanh activations. White “tanh” oval simply applies the tanh activation.

quantities of interest, $\mathbf{x} \in \mathbb{R}^{n_x}$ are input variables, and $f(.)$ is the transformation to the state. Depending on the application, $\mathbf{h}$ can be available (i.e., actually observed) in every time step t or only at specific observation times.

The repeating cells for a recurrent neural networks implement the function, $f(\mathbf{h}_{t-1}, \mathbf{x}_t)$ in Eq. 2.2, which defines the transformation applied to the states and inputs in every time step. Cells such as the ones illustrated in Fig. 2.4 are commonly found in data-driven applications. Figure 2.4a shows the simplest recurrent neural network cell, where a fully-connected dense layer (e.g., the perceptron) with a sigmoid activation function maps the inputs at time t and states at time $t-1$ into the states at time t. Figure 2.4b show two other popular architectures, the long short-term memory (LSTM) [64] and the gated recurrent unit (GRU) [65]. These architectures have extra elements (gates) to control the flow and update of the states through time and aims at improving the recurrent neural network generalization capability and training by mitigating the vanishing/exploding gradient problem [55]. Recurrent neural networks are trained in a very similar way of traditional neural networks. The inputs are fed forward for every time step through the cell. Then the loss value, calculated with the cell output and ground truth values, and its gradient is used to adjust the network weights, through the process called back-propagation through time [55].

CHAPTER 3: PHYSICS-INFORMED NEURAL NETWORK FOR ORDINARY DIFFERENTIAL EQUATIONS

In this chapter, we will focus on our hybrid physics-informed neural network implementation for ordinary differential equations. This is specially useful for problems where physics-informed models are available, but known to have predictive limitations due to model-form uncertainty or model-parameter uncertainty. We start by providing the background on recurrent neural networks and then discuss how we implement them for numerical integration.

First Order Ordinary Differential Equations

Consider the first order ordinary differential equation expressed in the form

$$\frac{dy}{dt} = f\left(\mathbf{x}(t), y, t\right) \tag{3.1}$$

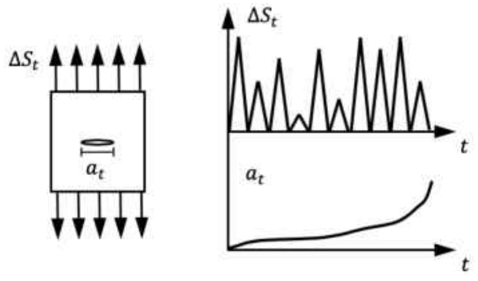
where $\mathbf{x}(t)$ are controllable inputs to the system, y is the output of interest, and t is time. The solution to Eq. (3.1) depends on initial conditions (y at $t = 0$), input data ($\mathbf{x}(t)$ known at different time steps), and the computational cost associated with the evaluation of $f(.)$.

Case Study: Fatigue Crack Propagation

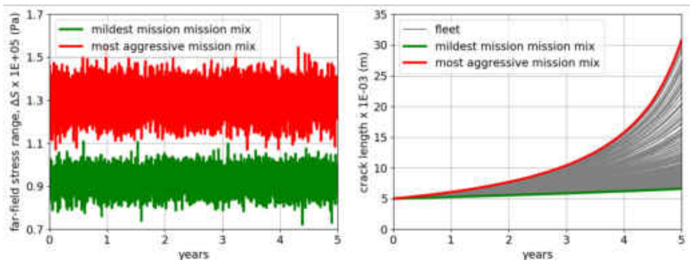
In this case study, we consider the tracking of low cycle fatigue damage. We are particularly interested in a control point that is monitored for a fleet of assets (e.g., compressors, aircraft, etc.). This control point sits on the center of large plate in which loads are applied perpendicularly to the crack plane. As depicted in Fig. 3.1a, under such circumstances, fatigue crack growth progresses following Paris law [66]

$$\frac{da}{dN} = C\left(\Delta K(t)\right)^m \quad \text{and} \quad \Delta K(t) = F\Delta S(t)\sqrt{\pi a(t)}, \tag{3.2}$$

where a is the fatigue crack length, C and m are material properties, ΔK is the stress intensity range, ΔS is the far-field cyclic stress time history, and F is a dimensionless function of geometry [67].



(a) Large plate with loads perpendicular to crack plane



(b) Snapshot of fleet-wide data (300 machines)

Figure 3.1: Fatigue crack propagation details. Crack growth is governed by Paris law, Eq. (3.2), a first order ordinary differential equation. The input is time history of far-field stress cyclic loads, ΔS, and the output is the fatigue crack length, a. In this case study, 300 aircraft are submitted to a wide range of loads (due to different missions and mission mixes). This explains the large variability in observed crack length after 5 years of operation.

We assume that the control point inspection occurs in regular intervals. Scheduled inspection of part of the fleet is adopted to reduce cost associated with it (mainly downtime, parts, and labor). As inspection data is gathered, the predictive models for fatigue damage are updated. In turn, the updated models can be used to guide the decision of which machines should be inspected next.

Figure 3.1b illustrates all the data used in this case study. There are 300 machines, each one accumulating 7,300 loading cycles. Not all machines are subjected to the same mission mix. In fact, the duty cycles can greatly vary, driving different fatigue damage accumulation rates throughout the fleet. In this case study, we consider that while the history of cyclic loads is known throughout the operation of the fleet, crack length history is not available. We divided the entire data set consisting of 300 machines into 60 assets used for training and 240 assets providing the test data sets. In real life, these numbers depend on the cost associated with inspection (grounding the aircraft implies in loss of revenue besides cost of the actual inspection). For the sake of this example, we observed 60 time histories of 7,300 data points each (total of 438,000 input points) and only 60 output observations. The test data consists of 240 time histories of 7,300 data points each (total of 1,752,000 input points) and no crack length observations. In order to highlight the benefits of the hybrid implementation, we use only 60 crack length observations after the entire load cycle regime. The fact that we directly implemented the governing equation in a recurrent neural network cell compensates for the number of points of available output data. Hence, for training procedures we only use the aforementioned 60 assets, while the data for the remaining 240 machines can be utilized as validation data set.

Computational Implementation

For the sake of this example, assume that the material is characterized by $C = 1.5 \times 10^{-11}$ and $m = 3.8$, $F = 1$, and that the initial crack length is $a_0 = 0.005$ meters. $\Delta S(t)$ is estimated either through structural health monitoring systems or high-fidelity finite element analysis together with cycle count methods (e.g., the rain flow method [68]). This way, the numerical method used to solve Eq. (3.2) hinges on the availability and computational cost associated with $\Delta S(t)$. In this example, let us assume that the far-field stresses are available in every cycle at very low computational cost (for example, the load cases are known and stress analysis is performed before hand).

Within folder `first_order_ode` of the repository available at [69], the interested reader will find all data files used in this case study. File a0.csv has the value for the initial crack length ($a_0 = 0.005$ meters) used throughout this case study. Files Stest.csv and Strain.csv contain the load histories for the fleet of 300 machines as well as the 60 machines used in the training of the physics-informed neural network. Files atest.csv and atrain.csv contain the fatigue crack length histories for the fleet as well as the 60 observed crack lengths used in the training of the physics-informed neural network.

We will then show how ΔK_t can be estimated through a multilayer perceptron (MLP), which works as a corrector on any poor estimation of either $\Delta S(t)$ or ΔK_t (should it have been implement through a physics-informed model). Therefore, we can simply use the Euler's forward method [70] (with unit time step) to obtain

$$a_n = a_0 + \sum_{t=1}^{n} \Delta a(\Delta S_t, a_{t-1}), \; , \tag{3.3}$$

$$\Delta a_t = C \Delta K_t^m \, , \quad \text{and} \quad \Delta K_t = \text{MLP}\left(\Delta S_t, a_{t-1}; \mathbf{w}, \mathbf{b}\right) \, , \tag{3.4}$$

where $\mathbf{w}$ and $\mathbf{b}$ are the trainable hyperparameters.

Similarly to regular neural networks, we use observed data to tune $\mathbf{w}$ and $\mathbf{b}$ by minimizing a loss function. Here we use the mean squared error:

$$\Lambda = \frac{1}{n} \left(\mathbf{a} - \hat{\mathbf{a}}\right)^T \left(\mathbf{a} - \hat{\mathbf{a}}\right), \tag{3.5}$$

where n is the number of observations, $\mathbf{a}$ are fatigue crack length observations, and $\hat{\mathbf{a}}$ are the predicted fatigue crack length using the hybrid physics-informed neural network.

Listing 3.1 lists all the necessary packages. Besides pandas and numpy for data importation and manipulation, we import a series of packages out of TensorFlow. We import Dense and RNN to

```python
# basic packages
import pandas as pd
import numpy as np

# keras essentials
from tensorflow.keras.layers import RNN, Dense, Layer
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import RMSprop

# tensorflow operators
from tensorflow.python.framework import tensor_shape
from tensorflow import float32, concat, convert_to_tensor
```

Listing 3.1: Import section for the Euler integration example.

leverage native multilayer perceptron and recurrent neural networks. We import `Sequential` and `Layer` so that we can specialize our model. We import `RMSprop` for hyperparameter optimization. Finally, the other operators are needed to move data to TensorFlow-friendly structures.

```python
class EulerIntegratorCell(Layer):
  def __init__(self, C, m, dKlayer, a0, units=1, **kwargs):
    super(EulerIntegratorCell, self).__init__(**kwargs)
    self.units = units
    self.C = C
    self.m = m
    self.a0 = a0
    self.dKlayer = dKlayer

    ...

  def call(self, inputs, states):

    ...

    x_d_tm1 = concat((inputs,a_tm1[0,:]),axis=1)
    dk_t    = self.dKlayer(x_d_tm1)
    da_t    = self.C * (dk_t ** self.m)
    a       = da_t + a_tm1[0, :]
    return a, [a]
```

Listing 3.2: Euler integrator cell.

Listing 3.2 shows the important snippets of implementation of the Euler integrator cell (to avoid clutter, we leave out the lines that are needed for data-type reinforcement). This is a class inherited from `Layers`, which is what TensorFlow recommends for implementing custom layers. The `__init__` method, constructor of the `EulerIntegratorCell`, assigns the constants C and m as well as the initial state a_0. This method also creates an attribute `dKlayer` to the object of `EulerIntegratorCell`. As we will detail later, this is an interesting feature that will essentially allow us to specify any model to `dKlayer`. Although `dKlayer` can be implemented using physics, as we discussed before, we will illustrate the case in which `dKlayer` is a multilayer perceptron. The `call` method effectively implements Eq. (3.4).

With regards to numerically integrating fatigue crack growth, we still have to implement Eq. (3.3). Here, we will use the TensorFlow native recurrent neural network class, `RNN`, to effectively march in time; and therefore, implement Eq. (3.3). Listing 3.3 details how we can use an object from the `EulerIntegratorCell` and couple it with `RNN` to create a model ready to be trained. The function `create_model` takes `C`, `m`, `a0`, and `dKLayer` so that an `EulerIntegratorCell` object can be instantiated. Additionally, it also takes `batch_input_shape` and `return_sequences`. The variable `batch_input_shape` is used within `EulerIntegratorCell` to reinforce the shape of the inputs. Although `batch_input_shape` is not directly specified in `EulerIntegratorCell`, it belongs to `**kwargs` and it will be consumed in the constructor of `Layer`.

```python
def create_model(C,m,dKlayer,a0,batch_input_shape,return_sequences):
    euler = EulerIntegratorCell(C=C,m=m,dKlayer=dKlayer,a0=a0,
            batch_input_shape=batch_input_shape,return_state=False)
    PINN = RNN(cell=euler,batch_input_shape=batch_input_shape,
            return_sequences=return_sequences,return_state=return_state)
    model = Sequential()
    model.add(PINN)
    model.compile(loss='mse',optimizer=RMSprop(1e-2))
  return model
```

Listing 3.3: Create model function for the Euler integration example.

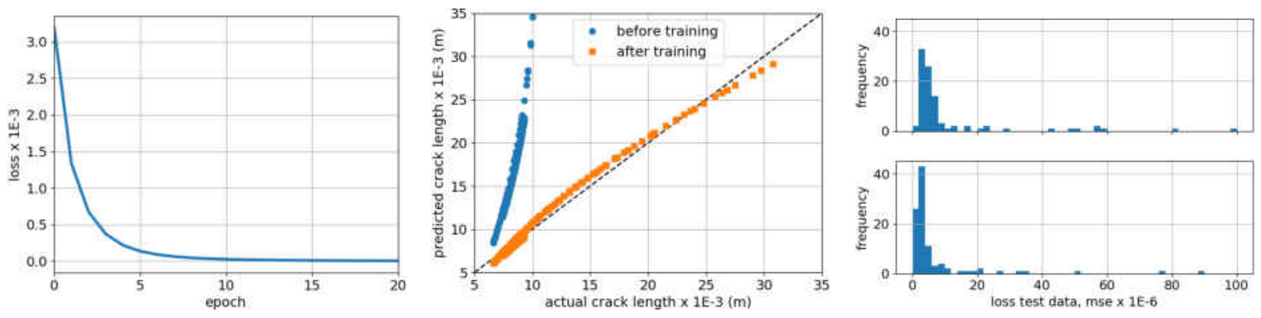
Equation 3.3 starts to be implemented when `PINN`, the object of class `RNN`, is instantiated. As a recurrent neural network, `PINN` has the ability to march through time and execute the `call` method of the `euler` object. Lines 10 to 14 of List. 3.3 are needed so that an optimizer and loss function are linked to the model that will be created. In this example, we use the mean square error (`'mse'`) as loss function and RMSprop as an optimizer.

```
 1 if __name__ == "__main__":
 2     # Paris law coefficients
 3     [C, m] = [1.5E-11, 3.8]
 4 
 5     # data
 6     Stest  = np.asarray(pd.read_csv('./data/Stest.csv'))[:,:,np.newaxis]
 7     Strain = np.asarray(pd.read_csv('./data/Strain.csv'))[:,:,np.newaxis]
 8     atrain = np.asarray(pd.read_csv('./data/atrain.csv'))
 9     a0     = np.asarray(pd.read_csv('./data/a0.csv'))[0,0]*np.ones((Strain.
    shape[0],1))
10 
11     # stress-intensity layer
12     dKlayer = Sequential()
13     dKlayer.add(Dense(5, input_shape=(2,), activation='tanh'))
14     dKlayer.add(Dense(1))
15 
16     ...
17 
18     # fitting physics-informed neural network
19     model = create_model(C=C, m=m, dKlayer=dKlayer,
20             a0=ops.convert_to_tensor(a0,dtype=float32),
21             batch_input_shape=Strain.shape)
22     aPred_before = model.predict_on_batch(Stest)[:,:]
23     model.fit(Strain, atrain, epochs=20, steps_per_epoch=1, verbose=1)
24     aPred = model.predict_on_batch(Stest)[:,:]
```

Listing 3.4: Training and predicting in the Euler integration example

With `EulerIntegratorCell` and `create_model` defined, we can proceed to training and predicting with the hybrid physics-informed neural network model. Listing 3.4 details how to build the main portion of the Python script. From line 2 to line 9, we are simply defining the material properties and loading the data. After that, we can create the `dKlayer` model. Within TensorFlow, `Sequential` is used to create models that will be stacks of several layers. `Dense` is used to define a layer of a neural network. Line 12 initializes `dKlayer` preparing it to receive the different layers in sequence. Line 13 adds the first layer with 5 neurons (and `tanh` as activation function). Line 14 adds the second layer with 1 neuron. Creating the hybrid physics-informed neural network model is as simple as calling `create_model`, as shown in line 19. As is, `model` is ready to be trained, which is done in line 23. For the sake of the example though, we can check the predictions at the training set before and after the training (lines 22 and 24, respectively). The fact that we have to slice the third dimension of the array with `[:,:]` is simply an artifact of TensorFlow.

The way the code is implemented, predictions are done by marching through time while integrating fatigue crack growth starting from `a0`. However, since we have set `return_sequences=False` (default in `create_model`), the predictions are returned only for the very last cycle. Setting that flag to `True` would change the behavior of the `predict_on_batch`, which would return the entire time series.



(a) Loss function convergence

(b) Predicted vs. actual crack length at the test set

(c) Mean square error histogram of on train and validation data (training repeated 100 times).

Figure 3.2: Euler integration results. After training is complete, the model-form uncertainty is greatly reduced. Trained model can be used directly for predictions outside the training set. We observe repeatability of results after repeating the training of the physics-informed neural network varying initialization of weights.

Figure 3.2 illustrates the results obtained when running the codes within folder `first_order_ode` available at [69]. Figure 3.2a shows the history of the loss function (mean square error) throughout the training. The loss converges rapidly within the first ten epochs and shows minor further convergence in the following ten epochs. We would like to point out that experienced Tensor-

Flow users could further customize the implementation to stop the hyperparameter optimization as loss function converges. Figure 3.2b shows the prediction against actual fatigue crack length at the last loading cycle for a test set (data points not used to train the physics-informed neural network). While results may vary from run-to-run, given that `RMSprop` implements a stochastic gradient descend algorithm, it is clear that the hybrid physics-informed neural network was able to learn the latent (hidden) stress intensity range model. Finally, we repeated the training of the proposed physics-informed neural network 100 times so that we can study the repeatability of results. Figure 3.2c shows the histograms of the mean squared error at both training and test sets. Most of the time, the mean squared error is bellow 20×10^{-6}(m)^2, while it was never above 100×10^{-6}(m)^2. Considering that the observed crack lengths are within 5×10^{-3} (m) to 35×10^{-3} (m), these values of mean square error are sufficiently small.

System of Second Order Ordinary Differential Equations

In this section, we will focus on our hybrid physics-informed neural network implementation of a system of second order ordinary differential equations. In the case study, we will highlight the useful aspect of system identification. This is when observed data is used to estimate parameters of the governing equations.

Consider the system of second order ordinary differential equation expressed in the form

$$\mathbf{P}(t)\frac{d^2\mathbf{y}}{dt^2} + \mathbf{Q}(t)\frac{d\mathbf{y}}{dt} + \mathbf{R}(t)\mathbf{y} = \mathbf{u}(t) \tag{3.6}$$

where $\mathbf{u}(t)$ are controllable inputs to the system, $\mathbf{y}$ are the outputs of interest, and t is time. The solution to Eq. (3.6) depends on initial conditions ($\mathbf{y}$ as well as $\frac{d\mathbf{y}}{dt}$ at $t = 0$), input data ($\mathbf{u}(t)$ known at different time steps).

Case Study: Forced Vibration of 2-Degree-of-Freedom System

In this case study, we consider the motion for two masses linked together springs and dashpots, as depicted in Fig. 3.3a. The number of degrees of freedom of a system is the number of independent coordinates necessary to define motion (equal to the number of masses in this case). Under such circumstances, the equations are obtained using Newton's second law

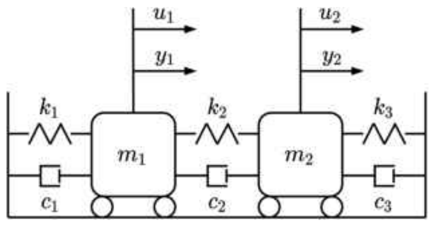
$$\begin{aligned} &\mathbf{M}\ddot{\mathbf{y}} + \mathbf{C}\dot{\mathbf{y}} + \mathbf{K}\mathbf{y} = \mathbf{u}, \text{ or alternatively} \\ &\ddot{\mathbf{y}} = f(\mathbf{u}, \dot{\mathbf{y}}, \mathbf{y}) = \mathbf{M}^{-1}\left(\mathbf{u} - \mathbf{C}\dot{\mathbf{y}} - \mathbf{K}\mathbf{y}\right), \end{aligned} \tag{3.7}$$

where:

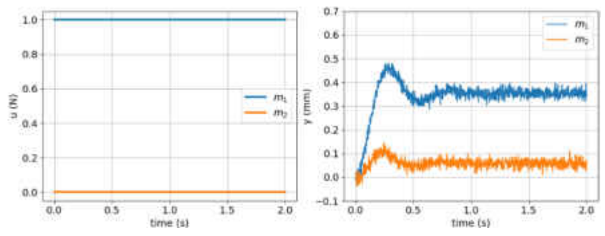
$$\mathbf{M} = \begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix}, \; \mathbf{C} = \begin{bmatrix} c_1 + c_2 & -c_2 \\ -c_2 & c_2 + c_3 \end{bmatrix}, \; \mathbf{K} = \begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 + k_3 \end{bmatrix}, \; \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \text{ and } \mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \tag{3.8}$$

We assume that while the masses and spring coefficients are known, the damping coefficients are not. Once these coefficients are estimated based on available data, the equations of motion can be used for predicting the mass displacements given any input conditions (useful for design of vibration control strategies, for example).

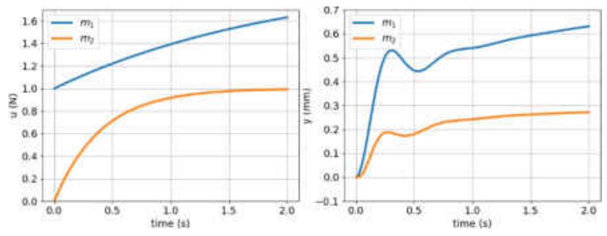
Figure 3.3b and 3.3c illustrate the data used in this case study. Here, we used $m_1 = 20$ (kg), $m_2 = 10$ (kg), $c_1 = 30$ (N.s/m), $c_2 = 5$ (N.s/m), $c_3 = 10$ (N.s/m), $k_1 = 2\times10^3$ (N/m), $k_2 = 1\times10^3$ (N/m), and $k_3 = 5\times10^3$ (N/m) to generate the data. On the training data, a constant force $u_1(t) = 1$ (N) is applied to mass m_1, while m_2 is let free. On the test data, time-varying forces are applied to both masses. The displacements of both masses are observed every 0.002 (s) for two seconds. The observed displacements of the training data are contaminated with Gaussian noise with zero mean and 1.5×10^{-5} standard deviation.



(a) Two degree of freedom system



(b) Input forces and displacements of training data



(c) Input forces and displacements of test data

Figure 3.3: Forced vibration details. Response of a two degree of freedom system is a function of input forces applied at the two masses. Training data is contaminated with Gaussian noise (emulating noise in sensor reading). Test data is significantly different from training data.

Computational Implementation

Within folder `second_order_ode` of the repository available at [69], the interested reader will find the training and test data in the `data.csv` and `data02.csv` files, respectively. The time stamp is given by column `t`. The input forces are given by columns `u1` and `u2`. The measured displacements are given by columns `yT1` and `yT2`. Finally, the actual (but unknown) displacements are given by columns `y1` and `y2`.

With defined initial conditions $\mathbf{y}(t = 0) = \mathbf{y}_0$ and $\dot{\mathbf{y}}(t = 0) = \dot{\mathbf{y}}_0$, we can use the classic Runge-

Kutta method [70, 71] to numerically integrate Eq. (3.7) over time set with time step h:

$$\begin{bmatrix}\dot{\mathbf{y}}_{n+1}\\ \mathbf{y}_{n+1}\end{bmatrix} = \begin{bmatrix}\dot{\mathbf{y}}_{n}\\ \mathbf{y}_{n}\end{bmatrix} + h\sum_i b_i \boldsymbol{\kappa}_i\,, \qquad \boldsymbol{\kappa}_i = \begin{bmatrix}\mathbf{k}_i\\ \bar{\mathbf{k}}_i\end{bmatrix}, \qquad \mathbf{k}_1 = f(\mathbf{u}_n, \dot{\mathbf{y}}_n, \mathbf{y}_n),\, \bar{\mathbf{k}}_1 = \mathbf{y}_n$$
$$\mathbf{k}_i = f\left(\mathbf{u}_{n+c_i h}, \dot{\mathbf{y}}_n + h\sum_j^{i-1} a_{ij}\mathbf{k}_j, \mathbf{y}_n + h\sum_j^{i-1} a_{ij}\bar{\mathbf{k}}_j\right), \qquad \bar{\mathbf{k}}_i = \mathbf{y}_n + h\sum_j^{i-1} a_{ij}\bar{\mathbf{k}}_j,$$
$$\mathbf{A} = \begin{bmatrix}0 & 0 & 0 & 0\\ 1/2 & 0 & 0 & 0\\ 0 & 1/2 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix}1/6\\ 1/3\\ 1/3\\ 1/6\end{bmatrix}, \qquad \mathbf{c} = \begin{bmatrix}0\\ 1/2\\ 1/2\\ 1\end{bmatrix}, \tag{3.9}$$

In this section, we will show how we can use observed data to tune specific coefficients in Eq. (3.7). Specifically, we will tune the damping coefficients c_1, c_2, and c_3 by minimizing the mean squared error:

$$\Lambda = \frac{1}{n}\left(\mathbf{y} - \hat{\mathbf{y}}\right)^T\left(\mathbf{y} - \hat{\mathbf{y}}\right), \tag{3.10}$$

where n is the number of observations, $\mathbf{y}$ are observed displacements, and $\hat{\mathbf{y}}$ are the displacements predicted using the physics-informed neural network.

We will use all the packages shown Listing 3.1, in addition to `linalg` imported from `tensorflow` (we did not show a separate listing to avoid clutter). Listing 3.5 shows the important snippets of implementation of the Runge-Kutta integrator cell (to avoid clutter, we leave out the lines that are needed for data-type reinforcement). The `__init__` method, constructor of the `RungeKuttaIntegratorCell`, assigns the mass, stiffness, and damping coefficient initial guesses, as well as the initial state and Runge-Kutta coefficients. The `call` method effectively implements Eq. (3.9) while the `_fun` method implements Eq. (3.7).

```python
class RungeKuttaIntegratorCell(Layer):
    def __init__(self, m, c, k, dt, initial_state, **kwargs):
        super(RungeKuttaIntegratorCell, self).__init__(**kwargs)
        self.Minv = linalg.inv(np.diag(m))
        self._c   = c
        self.K    = self._getCKmatrix(k)
        self.A  = np.array([0., 0.5, 0.5, 1.0], dtype='float32')
        self.B  = np.array([[1/6, 2/6, 2/6, 1/6]], dtype='float32')
        self.dt = dt
        ...

    def build(self, input_shape, **kwargs):
        self.kernel = self.add_weight("C",shape=self._c.shape,
            trainable=True, initializer=lambda shape,dtype: self._c,
            **kwargs)
        self.built  = True

    def call(self, inputs, states):
        C    = self._getCKmatrix(self.kernel)
        y    = states[0][:,:2]
        ydot = states[0][:,2:]

        yddoti = self._fun(self.Minv, self.K, C, inputs, y, ydot)
        yi     = y + self.A[0] * ydot * self.dt
        ydoti  = ydot + self.A[0] * yddoti * self.dt
        fn     = self._fun(self.Minv, self.K, C, inputs, yi, ydoti)
        for j in range(1,4):
            yn    = y + self.A[j] * ydot * self.dt
            ydotn = ydot + self.A[j] * yddoti * self.dt
            ydoti = concat([ydoti, ydotn], axis=0)
            fn    = concat([fn,self._fun(self.Minv,self.K,C,inputs,yn,ydotn)
],axis=0)

        y    = y + linalg.matmul(self.B, ydoti) * self.dt
        ydot = ydot + linalg.matmul(self.B, fn) * self.dt
        return y, [concat(([y, ydot]), axis=-1)]

    def _fun(self, Minv, K, C, u, y, ydot):
        return linalg.matmul(u - linalg.matmul(ydot, C, transpose_b=True) -
linalg.matmul(y, K, transpose_b=True), Minv, transpose_b=True)
        ...
```

Listing 3.5: Runge-Kutta integrator cell.

Listing 3.6 details how we use objects from `RungeKuttaIntegratorCell` and `RNN` to create a model ready to be trained. The function `create_model` takes `m`, `c`, and `k` arrays, `dt`, `initial_state`, `batch_input_shape` and `return_sequences` so that a `RungeKuttaIntegratorCell` object is instantiated. Parameter `batch_input_shape` is used within `RungeKuttaIntegratorCell` to reinforce the shape of the inputs (although it is not directly specified in `RungeKuttaIntegratorCell`, it belongs to `**kwargs` and it will be consumed in the constructor of `Layer`).

Similarly to the Euler example, we will use the TensorFlow native recurrent neural network class, `RNN`, to effectively march in time. The march through time portion of Eq. (3.7) starts to be implemented when `PINN`, the object of class `RNN`, is instantiated. As a recurrent neural network, `PINN` has the ability to march through time and execute the `call` method of the `rkCell` object. Lines 9 to 11 of List. 3.6 are needed so that an optimizer and loss function are linked to the model that will be created. In this example, we use the mean square error (`'mse'`) as loss function and RMSprop as an optimizer.

```
1  def create_model(m,c,k,dt,initial_state,batch_input_shape,
2                  return_sequences=True, unroll=False):
3      rkCell = RungeKuttaIntegratorCell(m=m,c=c,k=k,dt=dt,
4              initial_state=initial_state)
5      PINN = RNN(cell=rkCell,batch_input_shape=batch_input_shape,
6              return_sequences=return_sequences,
7              return_state=False,unroll=unroll)
8      model = Sequential()
9      model.add(PINN)
10     model.compile(loss='mse',optimizer=RMSprop(1e4),metrics=['mae'])
11     return model
```

Listing 3.6: Create model function for the Runge-Kutta integration example.

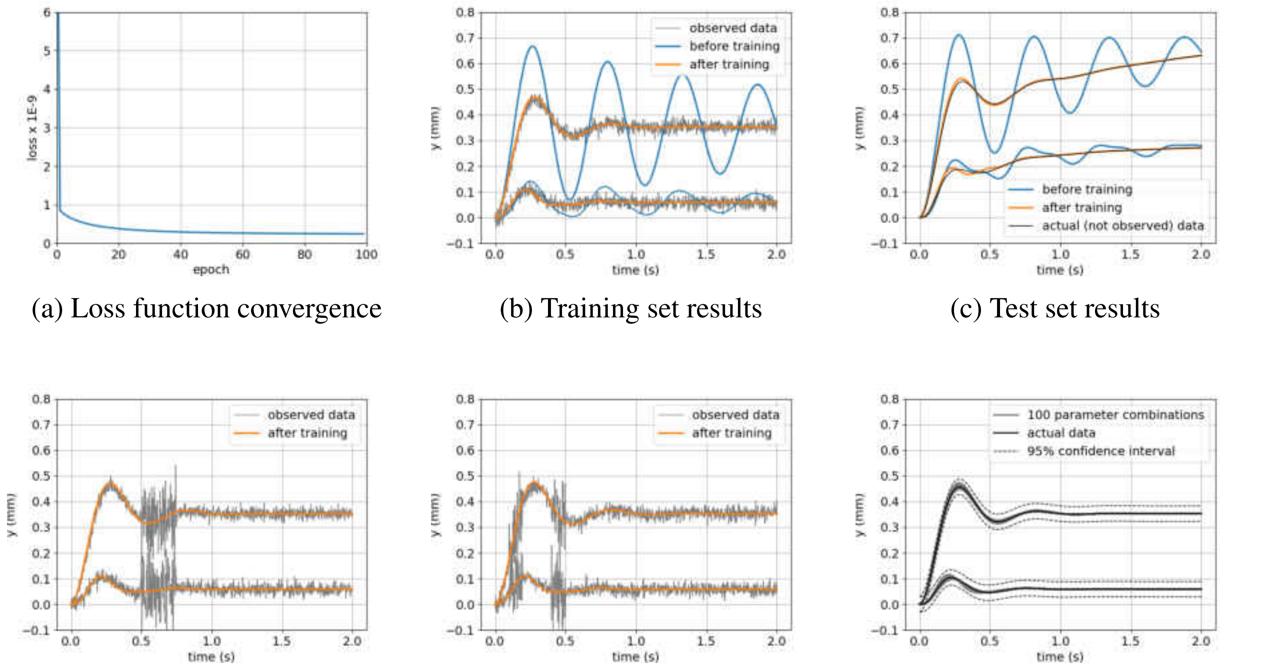
Listing 3.7 details how to build the main portion of the Python script. From line 2 to line 5, we are simply defining the masses, spring coefficients (which are assumed to be known), as well as damping coefficients, which are unknown and will be fitted using observed data (here, values only represent an initial guess for the hyperparameter optimization). Creating the hybrid physics-informed neural network model is as simple as calling `create_model`, as shown in line 16. As is, `model` is ready to be trained, which is done in line 19. or the sake of the example though, we can check the predictions at the training set before and after the training (lines 18 and 20, respectively).

```
if __name__ == "__main__":
    # masses, spring coefficients, and damping coefficients
    m = np.array([20.0, 10.0],dtype='float32')
    c = np.array([10.0, 10.0, 10.0],dtype='float32') # initial guess
    k = np.array([2e3, 1e3, 5e3],dtype='float32')

    # data
    df = pd.read_csv('./data/data.csv')
    t  = df[['t']].values
    dt = (t[1] - t[0])[0]
    utrain = df[['u0','u1']].values[np.newaxis,:,:]
    ytrain = df[['yT0','yT1']].values[np.newaxis,:,:]

    # fitting physics-informed neural network
    initial_state = np.zeros((1,2*len(m),),dtype='float32')
    model = create_model(m,c,k,dt,initial_state=initial_state,
        batch_input_shape=utrain.shape)
    yPred_before = model.predict_on_batch(utrain)[0,:,:]
    model.fit(utrain, ytrain, epochs=100, steps_per_epoch=1, verbose=1)
    yPred = model.predict_on_batch(utrain)[0,:,:]
```
Listing 3.7: Training and predicting in the Runge-Kutta integration example

Figure 3.4 illustrates the results obtained when running the codes within folder `second_order_ode` available at [69]. Figure 3.4a shows the history of the loss function (mean square error) throughout the training. Figures 3.4b and 3.4c show the prediction against actual displacements. Similarly to the Euler case study, results may vary from run-to-run, depending on the initial guess for c_1, c_2, and c_3 as well as performance of `RMSprop`. The loss converges rapidly within 20 epochs and only marginally further improves after 40 epochs. As illustrated in Fig. 3.4b, the predictions converge to the observations, filtering the noise in the data. Figure 3.4c shows that the model parameters identified after training the model allowed for accurate predictions on the test set. In order to further evaluate the performance of the model, we created contaminated training data sets where we emulate the case that sensors used to read the output displacement exhibit a burst of high noise levels at different points in the time series. For example, Fig. 3.4d illustrates the case in which the burst of high noise level happens between 0.5 (s) and 0.75 (s); while in Fig. 3.4e, this data corruption happened at two different time periods (0.1 to 0.2 (s) and 0.4 to 0.5 (s)). In both cases, model parameters identified after training the model allowed for accurate predictions.



(a) Loss function convergence

(b) Training set results

(c) Test set results

(d) Prediction at first contaminated training set

(e) Prediction at second contaminated training set

(f) Damping coefficient variation. Confidence interval based on Gaussian noise.

Figure 3.4: Runge-Kutta integration results. After training, damping coefficients are identified (Tab. 3.1). Model can be used in test cases that are completely different from training. Due to nature of physics that governs the problem, responses are less sensitive to coefficients c_2 and c_3, when compared to c_1. Nevertheless, model identification is successful even when noise level varies throughout training data.

Noise in the data imposes a challenge for model parameter identification. Table 3.1 lists the identified parameters for the separate model training runs with and without the bursts of corrupted data. As expected, c_1 is easier to identify, since it is connected between the wall and m_1, which is twice as large as m_2. On top of that, the force is applied in m_1. In this particular example, the outputs show low sensitivity to c_2 and c_3. Figure 3.4f show a comparison between the actual training data (in the form of mean and 95% confidence interval) and the predicted curves when $70 \leq c_2 \leq 110$

and $15 \leq c_3 \leq 120$. Despite the apparently large ranges for c_2 and c_3, their influence in the variation of the predicted output is still smaller than the noise in the data.

Table 3.1: Identified damping coefficients. Actual values for the coefficients are $c_1 = 100.0$, $c_2 = 110.0$, and $c_3 = 120.0$. Due to nature of physics that governs the problem, responses are less sensitive to coefficients c_2 and c_3, when compared to c_1 (Fig. 3.4f).

Noise in observed output	c_1	c_2	c_3
Gaussian	115.1	71.6	16.7
Gaussian with single burst of high contamination	113.2	70.0	17.1
Gaussian with double burst of high contamination	109.2	70.7	15.3

CHAPTER 4: FLEET PROGNOSIS WITH HYBRID MODELS

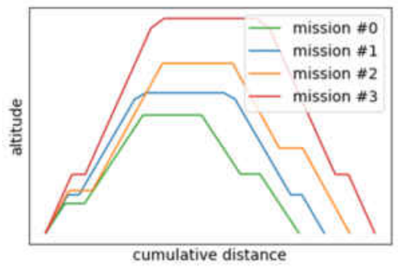
First we discuss the specifics of fatigue crack growth at a specific control point and the extension to the fleet of aircraft. Then, we present two scenarios for fatigue crack growth estimation of aircraft fuselage panels and prediction at a fleet of aircraft. In the first scenario, we consider that the control point in the fuselage panel is instrumented and continuously monitored through dedicated structural health monitoring sensors (e.g., comparative vacuum monitoring [72], fiber Bragg grating sensors [73], etc.). In the second scenario, we consider the control point in the fuselage panel is inspected in regular intervals through non-destructive evaluation approaches (e.g., Eddy current [74], ultrasound [75], dye penetrant inspection [76], etc.).

These applications impose two major challenges for recurrent neural networks: 1) the sequences are very large (thousands of steps), and 2) output values are known at the beginning of the sequence but are observed only at few time stamps throughout the sequence. The large sequences can lead to a significant increase in the norm of the gradients during training, which can harm the learning process. Moreover, by having only a few observations throughout the sequence, the long-term components can grow or decrease exponentially (exploding/vanishing gradient). This fast saturation makes it very hard for the model to learn the correlation between the observations. The interested reader is referred to the work of Pascanu et al. [77] and Sutskever [78] for further discussion on the difficulties of training recurrent neural networks under such circumstances.

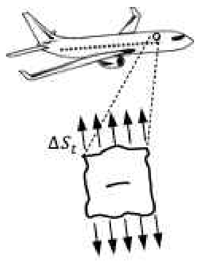
Fleet of aircraft and fuselage panel control point

When airline companies operate a fleet of a particular aircraft model, they usually adopt a series of operation and maintenance procedures that maximize the use of their fleet under economical and safety considerations. For example, these companies rotate their aircraft fleet through different routes following specific mission mixes. This way, no single aircraft is always exposed to the most aggressive (or mildest) routes, which helps managing useful lives of critical components. Here, we assume an aircraft model designed to fly the four hypothetical missions shown in Fig. 4.1a) and consider the mission mixes detailed in Tab. 4.1. We assume the airline company has 300 aircraft allocated to each mission mix. Each aircraft is assigned a fixed percentage of flights for each mission that composes the mission mix. These percentages vary uniformly from 0% to 100%. Therefore, within the fleet flying mission mix A, for example, there is one aircraft flying 0% of mission $\#0$ and 100% of mission $\#3$, there is another aircraft flying 1% of mission $\#0$ and 99% of mission $\#3$, there is yet another aircraft flying 2% of mission $\#0$ and 98% of mission $\#3$, and so forth. The same logic applies to the other mission mixes.

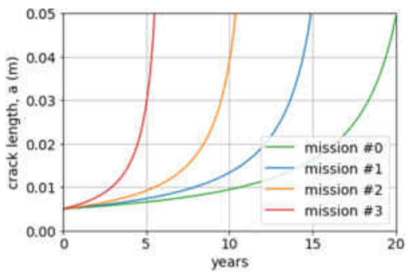
We chose these four missions as an effort to illustrate the different conditions the fleet of aircraft usually experience in commercial aviation. In reality though, the number of missions and the number of mission mixes depends on how operators decide to manage their fleets. This way, we synthetically created data for a fleet of 300 aircraft.



(a) Flight profile for different missions.



(b) Control point on the aircraft fuselage.



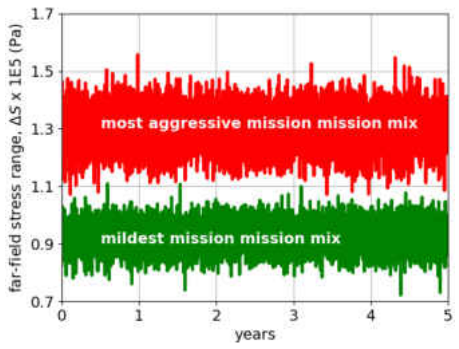
(c) Cumulative damage over time.

Figure 4.1: Cumulative damage over time for control point on the aircraft fuselage as a function of mission profile (assuming aircraft flies four missions per day).

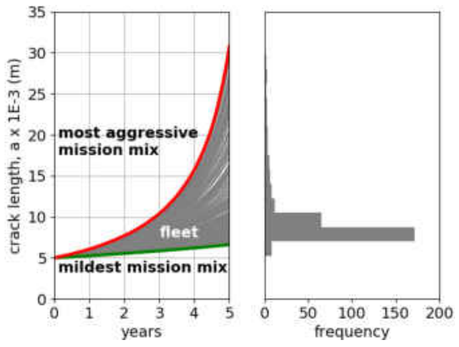
Table 4.1: Missions mix details. Percentage flights for each mission vary from 0% to 100%.

Mission mix	Mission (load in KPa)			
	#0 (92.5)	#1 (100)	#2 (110)	#3 (130)
A	✓			✓
B		✓	✓	
C		✓		✓

We consider the control point on an aircraft fuselage illustrated in Fig. 4.1b (crack in infinite plate) and assume that fatigue damage accumulates throughout the useful life, as illustrated in Fig. 4.1c. For the sake of this example, we assumed that the initial and the maximum allowable crack lengths are $a_0 = 0.005$ m and $a_{max} = 0.05$ m, respectively. The metal alloy is characterized by the following Paris law constants: $C = 1.5 \times 10^{-11}$, $m = 3.8$.



(a) Missions histories for two different aircraft.



(b) Fatigue crack length history for the fleet (300 aircraft).

Figure 4.2: Snapshot of synthetic data.

Figure 4.2 illustrates part of the data used here. Figure 4.2a shows the complete mission history in terms of far-field stresses for the aircraft flying the most aggressive and most mild mission mixes of the fleet (for the entire fleet, the far-field stress time histories follow the mixes described in Tab. 4.1). Figure 4.2b shows how the crack length time histories can be different across the entire fleet and highlights the two extreme cases (most aggressive and most mild mission mixes of the fleet) as well as the fleet-wide crack length distribution at the 5th year. For the sake of this study, we consider that the history of cyclic loads is known throughout the operation of the fleet (i.e., the data shown in Fig. 4.2a is observed). On the other hand, the crack length history is only partially known (i.e., data shown in Fig. 4.2b) and the availability of the information depends on the adopted monitoring/inspection strategy.

Scenario I: continuous monitoring of control point

As previously mentioned, in the first scenario, we assume that the control point is instrumented and continuously monitored through dedicated structural health monitoring sensors (e.g., comparative vacuum monitoring and fiber Bragg grating sensors, etc.). In practical applications, airline companies could limit the number of monitored aircraft due to cost associated with the structural health monitoring system (including its own maintenance). In such cases, data gathered on part of the fleet is used to build predictive models for the entire fleet.

Table 4.2: Inspection periods with its number of time steps and total data points used for training

Periodicity	Yearly	Monthly	Weekly	Daily
Number of time steps (per aircraft)	5	60	260	1825
Number of data points (60 aircraft)	300	3600	15600	109,500

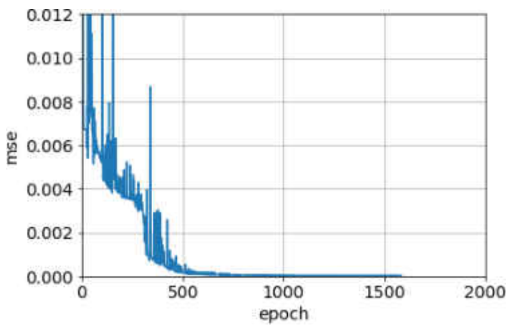
We arbitrarily assume the dedicated structural health monitoring sensor were installed on 60 aircraft of the fleet. This is equivalent to 20% of the fleet and represents a scenario in which the airline companies are willing to instrument a significant portion of the fleet. It should also allow us to study continuous monitoring without having to instrument the entire fleet or having only very few instrumented aircraft. As detailed in Tab. 4.2, we studied different rates in which sensor data is acquired (from data collection as sparse as once a year to as refined as once a day). As mentioned before, we also considered the case of weekly and daily crack length observation.

First we evaluated the performance of purely data-driven recurrent neural networks. We used the long short-term memory and and gated recurrent unit (Fig. 2.4b) architectures. Multiple configurations were tested varying the number of layers (stacked cells) and units (with the same number of neurons per layer), as detailed in Tab. 4.3. Every one of the nine configurations was used for each inspection period (yearly, monthly, weekly, and daily) totaling 72 distinct models (36 LSTM and 36 GRU). The training of these neural networks was performed using the mean square error (MSE) loss function and the Adam optimizer [79]. The different model architectures converged at slightly different epochs, all returning the best results before 1000 epochs. Nevertheless, the training was carried over to 2000 epochs to all models to ensure no further improvement would occur.

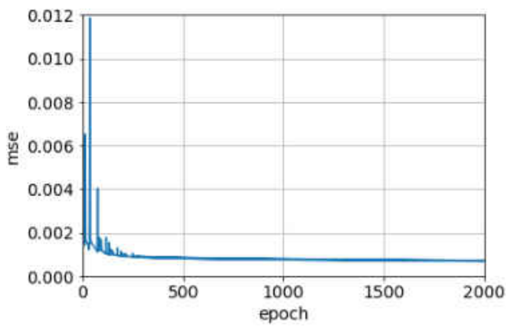
Table 4.3: Designs based for long short-term memory (LSTM) and gated recurrent unit (GRU) networks.

	Configuration								
Layers	1	1	1	2	2	2	4	4	4
Neurons	16	32	64	16	32	64	16	32	64
	Number of parameters								
LSTM	1168	4384	16960	3280	12704	49984	7504	29344	116032
GRU	928	3392	12928	2560	9728	37888	5824	22400	87808

As illustrated in Fig. 4.3, the loss function of most of the models converged to small values (althougth their rate of convergence can greatly differ). Since we are using the Adam optimizer, we see high oscillation early in the optimization process is a manifestation of the learning rate adjustment, which is followed by a rapid convergence midway, and then stagnation towards the end the optimization. After training with the sub-fleet of 60 aircraft, all models were validated against the entire fleet (300 aircraft).



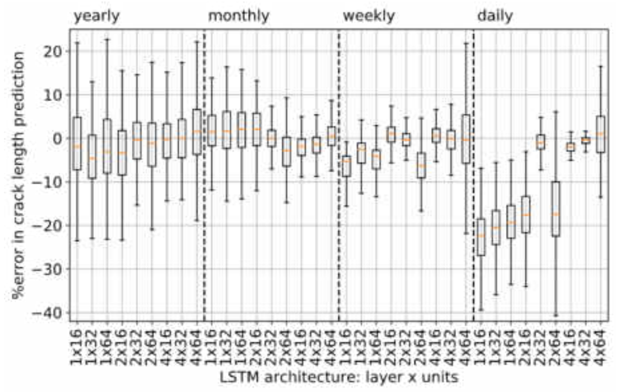
(a) LSTM, 4×32, daily.



(b) GRU, 4×32, daily.

Figure 4.3: Example of loss function history throughout the training of the long short-term memory (LSTM) and gated recurrent unit (GRU) networks.

Figure 4.4 shows the percent prediction errors of the recurrent neural network models at the end of the 5th year for all 300 aircraft when crack length is observed yearly, monthly, weekly, and daily. This detailed look at the prediction accuracy reveals that the number of observations strongly contributes to overall prediction accuracy. Not surprisingly, the more observations used in training the more accurate the model predictions are. Although the boxplots show that some architectures



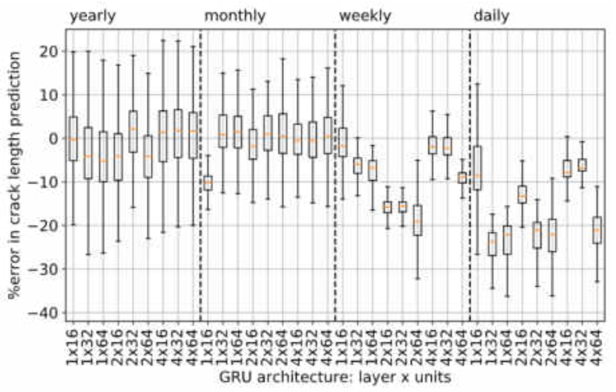
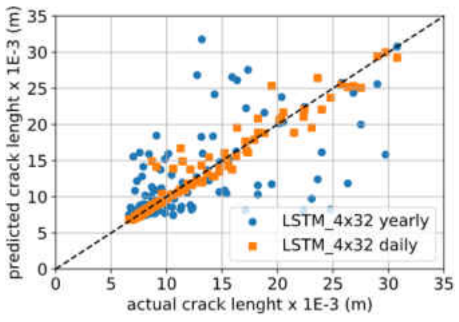


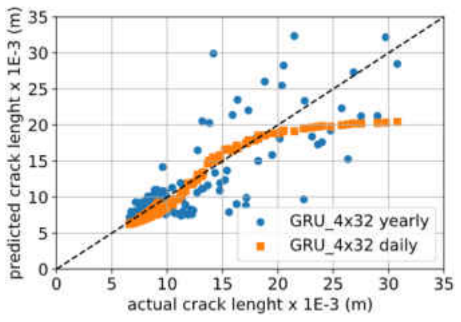
Figure 4.4: Boxplot of percent error in crack length prediction for the long short-term memory (LSTM) and gated recurrent unit (GRU) networks at the 5th year for entire fleet (300 aircraft). $\%error = 100 \times \frac{a_{PRED} - a_{actual}}{a_{actual}}$.

would outperform others, there is no clear trend with regards to complexity (i.e., more complex models outperforming simpler models up to a point). This indicates that the result is likely to be dependent on the specific training of the neural network (through a combination of random initialization of weights, and optimization parameters such as optimization algorithm, learning rate, number of epochs, etc.).

Figure 4.5 shows how the predictions at the end of the 5th year for all 300 aircraft compare with actual crack lengths. Interestingly, Figs. 4.5a and 4.5b show that both the LSTM- and the GRU-based recurrent neural networks have similarly poor performance when trained with yearly observations. The performance improves when these models are trained with daily observations; although there is still considerable prediction error across the crack length range (the GRU-based seems to exhibit a bias towards the high crack length).



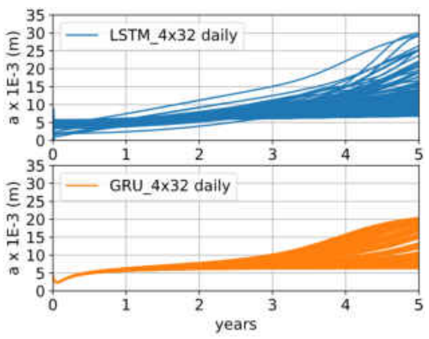
(a) Long-short term memory (LSTM).



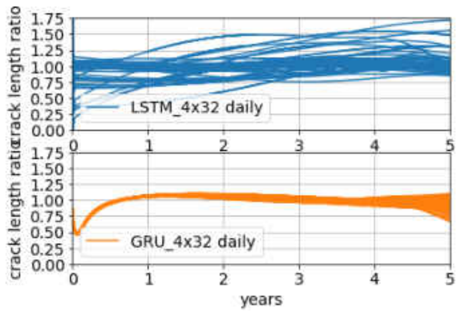
(b) Gated recurrent unit (GRU).

Figure 4.5: LSTM and GRU predictions versus actual crack length at the 5th year for entire fleet (300 aircraft).

Figure 4.6 illustrates the model predictions up to the end of the 5th year for all 300 aircraft. Figure 4.6a shows the time histories for the actual crack length and the model predictions coming from one LSTM- and one GRU-based neural network. Figures 4.5 and 4.6a clearly show that the crack length trends might be captured but the shape of the curve is poorly approximated. Figure 4.6b



(a) Crack length histories.



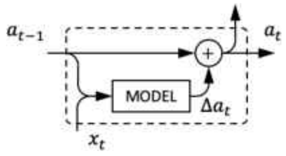
(b) Ratio between predicted and actual crack length.

Figure 4.6: Actual and LSTM- and GRU-predicted crack length over time for the entire fleet (300 aircraft). LSTM and GRU stand for long short-term memory and gated recurrent unit, respectively.

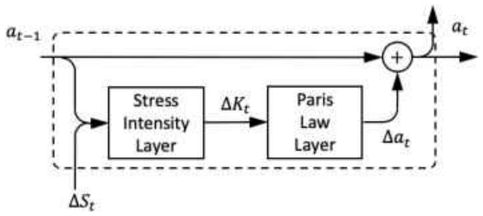
illustrates the ratio between the predicted and actual crack lengths. The ratio being close to one is a good indication of prediction accuracy. For both LSTM- and GRU-based neural networks, the predictions are mostly within $\pm 25\%$ (except for some predictions out of the LSTM-based neural network, which can overestimate the final crack lengths by as much as 75%).

We also tested the performance of the proposed hybrid physics-informed neural network when there is continuous monitoring of the control point. We built the model using multi-layer perceptrons as stress intensity layer in series with a physics-based Paris law layer (as illustrated in Fig. 4.7c). Table 4.4 details the multi-layer perceptron designs considered in this study. We varied the number of layers and neurons within each layer as well as the activation functions. We used the linear, hyperbolic tangent (tanh), and the exponential linear unit (elu) activation functions given as follows

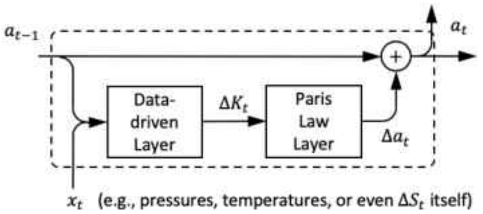
$$\text{linear}(z) = z, \; \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \text{ and } \text{elu}(z) = \begin{cases} z & \text{when } z > 0 \\ e^z - 1 & \text{otherwise.} \end{cases} \tag{4.1}$$



(a) Cumulative damage RNN cell.



(b) Fully physics-informed cell.



(c) Hybrid physics-informed neural network cell.

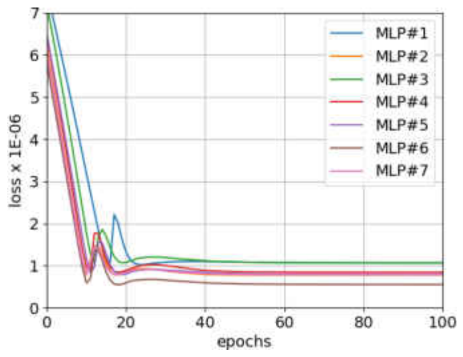
Figure 4.7: Cumulative damage and examples of RNN cells for crack growth. The stress intensity range layer implements $\Delta K_t = F\Delta S_t\sqrt{\pi a_{t-1}}$ and the Paris law layer implements $\Delta a_t = C\Delta K_t^m$.

Table 4.4: Multi-layer perceptron (MLP) configurations used to model the stress intensity range.

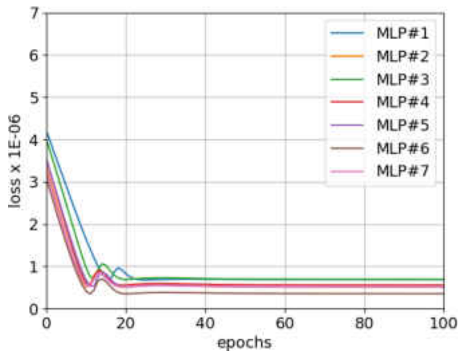
Layer #	Number of neurons / activation function						
	MLP #1	MLP #2	MLP #3	MLP #4	MLP #5	MLP #6	MLP #7
0	5 / tanh	10 / elu	10 / tanh	10 / tanh	20 / tanh	20 / tanh	40 / tanh
1	1 / linear	5 / elu	5 / elu	5 / tanh	10 / elu	10 / elu	20 / elu
2		1 / linear	1 / linear	1 / linear	5 / elu	5 / tanh	10 / elu
3					1 / linear	1 / linear	5 / tanh
4							1 / linear
Parameters	21	91	91	91	331	331	1,211

The training of these neural networks was performed using the mean square error loss function and the Adamax optimizer [79]. Given that the hybrid physics-informed neural network converged much faster (all models converged before 50 epochs), the training was carried over to 100 epochs to ensure no further improvement would still occur.

Similarly to what happened with the data-driven architectures (see Fig. 4.3), Fig. 4.8 illustrates the loss function oscillation early in the optimization process followed by convergence and stagnation towards the end the optimization. Figure 4.8 also shows that the different multi-layer perceptrons



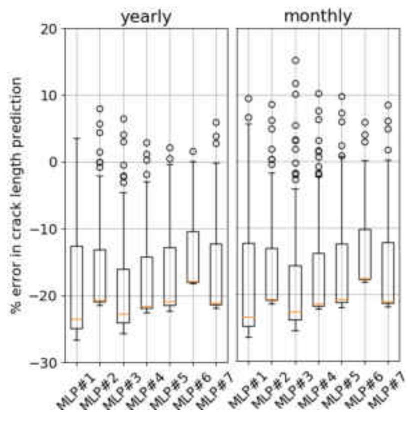
(a) Yearly observations.



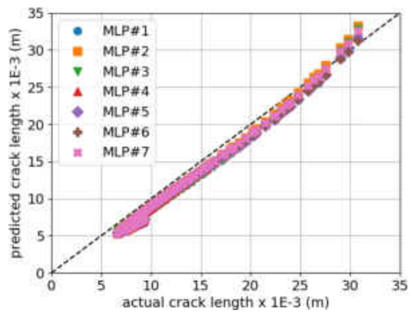
(b) Monthly observations.

Figure 4.8: Example of loss function history throughout the training of the physics-informed neural networks with continuous monitoring of the control point.

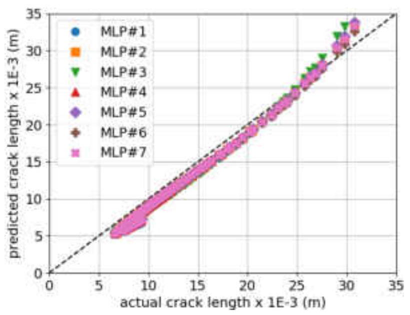
converged to roughly the same loss function values at the end of the training process. Figure 4.9a shows the percent prediction errors of the recurrent neural network models at the end of the 5th year for all 300 aircraft when crack length is observed yearly and monthly. The main advantage of using physics-informed neural networks is reducing the need for training points. In this case, the costly part is the acquisition of crack length observations. One can argue that a monitoring system that off-loads data yearly or monthly is cheaper to operate and maintain when compared to one that is expected to produce data on a daily basis, for example. Therefore, we will only show results for yearly and monthly crack length observations. Visual comparison shows that all the seven physics-informed neural networks had percent errors within the same order of magnitude. This is true even when models are trained with yearly observations, which confirms that the physics-informed layer compensates for the reduced number of observations. From an airline company perspective, the motivation behind the reduction in the number of training points is the cost associated with continuously monitoring the aircraft in the fleet. In this case study, the costly part is the acquisition of crack length observations. Therefore, it is expected that a monitoring system that off-loads data yearly or monthly is more economical to operate and maintain when compared to one that is



(a) Boxplot of percent error in crack length prediction.



(b) Yearly observations.



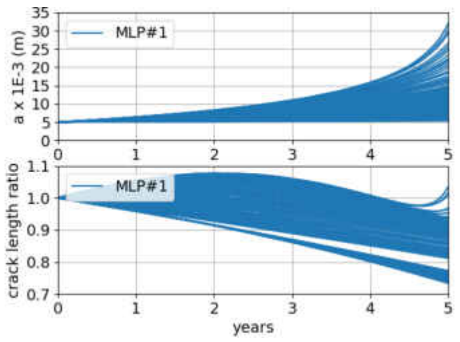
(c) Monthly observations.

Figure 4.9: Physics-informed neural network predictions versus actual crack length at the 5th year for entire fleet (300 aircraft). $\%error = 100 \times \frac{a_{PRED} - a_{actual}}{a_{actual}}$

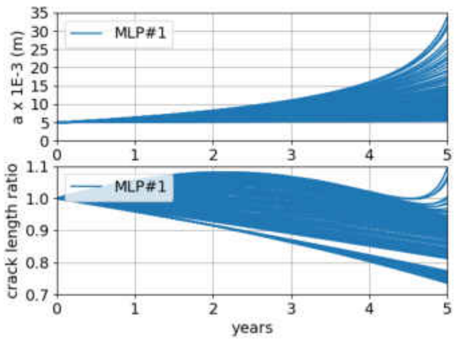
expected to produce data on a daily basis, for example.

Figure 4.9 shows how the predictions at the end of the 5th year for all 300 aircraft compare with actual crack lengths. Interestingly, Figs. 4.9b and 4.9c confirm the robustness of the physics-informed neural networks when trained with yearly observations. We also see that the relatively large percent errors (let us say above 10%) are likely coming from aircraft exhibiting small crack lengths at the end of the 5th year. On the basis of the number of training points and prediction errors, the physics-informed neural networks outperform the data-driven models for this application.

Finally, Fig. 4.10b illustrates the model predictions up to the end of the 5th year for all 300 aircraft for the multi-layer perceptron #1 (detailed in Tab. 4.4). As opposed to Fig. 4.6, Fig. 4.10b shows that the crack length trends as well as the curve over time are well approximated (that is attributed to the physics-informed layer). The models are better suited for tracking crack length than the data-driven counterparts (i.e., the LSTM- and one GRU-based neural networks).



(a) Yearly observations.



(b) Monthly observations.

Figure 4.10: Actual and physics-informed neural network predicted crack length over time.

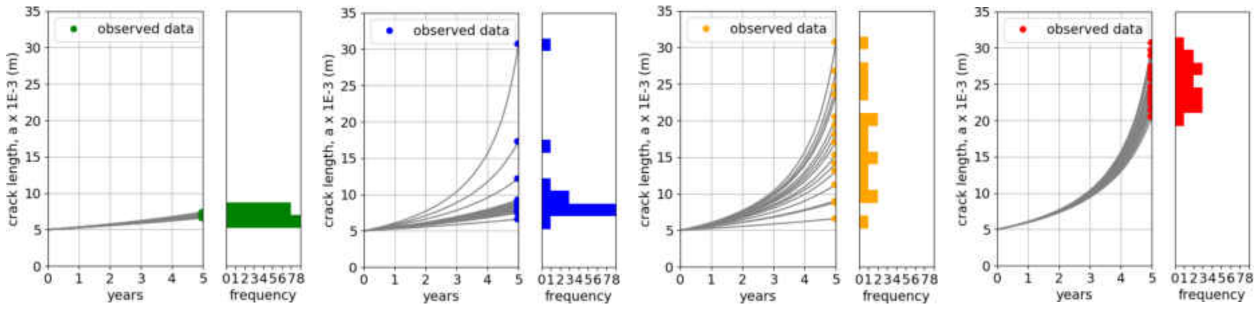
Scenario II: scheduled inspection of control point

As previously mentioned, in the second scenario, we also consider the case in which the control point in the fuselage panel is inspected in regular intervals through non-destructive evaluation approaches (e.g., Eddy current, ultrasound, dye penetrant inspection, etc.). Due to cost associated with inspection (mainly downtime, parts, and labor), it is customary to perform inspection in predefined intervals (which might vary from control point to control point). Usually, aircraft are inspected in batches to avoid grounding the entire fleet. In such case, data gathered on part of the fleet is used to build predictive models for the entire fleet. These predictive models can be used to guide the decision of which aircraft should be inspected next.

Table 4.5 details the hypothetical cases for selecting the aircraft out of fleet for inspection. Figure 4.11 highlights the 15 observed crack lengths at the end of the 5th year for cases $\#1$ to $\#3$ of Tab. 4.5. In the training of the recurrent neural networks, the inputs are always observed (i.e., the full far-field stress range time history is observed). However, the output is only observed at inspection. At a rate of 4 flights per day, in a period of 5 years, this means that we observed 5 to 60 time histories of 7,300 data points each (total of 36,500 to 438,000 input points) and only

Table 4.5: Scenarios for inspection data.

Distribution of observed crack length	Inspected aircraft			
	5	15	30	60
Case #1 - Biased towards small crack lengths		✓		
Case #2 - Following true distribution of crack length		✓		
Case #3 - Uniform coverage of crack lengths	✓	✓	✓	✓
Case #4 - Biased towards large crack lengths		✓		

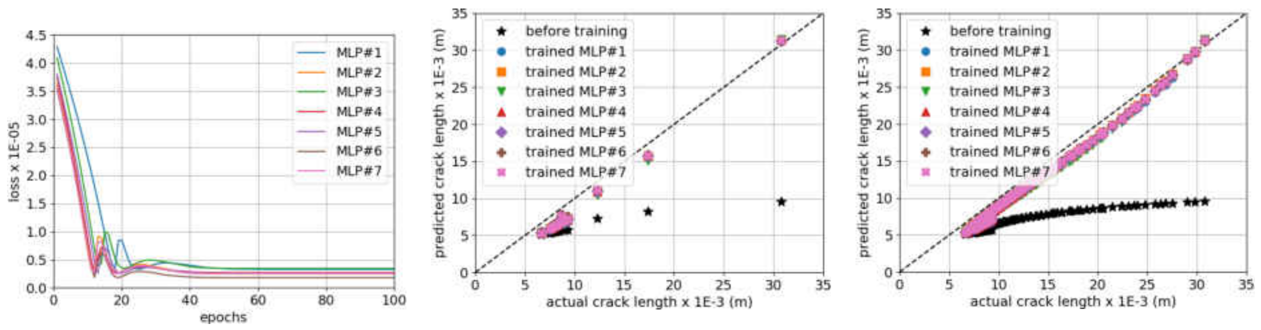


(a) Case #1. (b) Case #2. (c) Case #3. (d) Case #4.

Figure 4.11: Fatigue crack length history and observations (15 aircraft) at the 5th year. Details about each case are found in Tab. 4.5.

5 to 60 output observations. Here, the intent is to study the influence of number and distribution of inspections (output observations) in the training of the neural network. In all cases, inspection is assumed to take place at the 5th year of operation. Similarly to what would happen in real life, the first inspection results are used to train the models, which will be used to make crack length prediction across the entire fleet. Based on the performance results of data-driven and physics-informed neural networks in scenario I, we decided to focus this study only on the physics-informed neural networks. The poor performance of the purely data-driven neural networks as the number of training points gets reduced indicates they are not suitable for scenario II.

We first study the impact of the multi-layer perceptron design in the training and validation of the physics-informed recurrent neural networks. We use data from 15 aircraft in which there is uniform coverage of the crack lengths (case #3 from Tab. 4.5 shown in Fig. 4.11c). The mean square error has fast convergence throughout training, as shown in Fig. 4.12a. Figure 4.12b shows the predictions at the end of the 5th year for the training set (15 inspected aircraft) before and after training. Figure 4.12c shows the predictions at the end of the 5th year for the entire fleet (300 aircraft) before and after training. In both cases, the model initially tends to under-predict the large crack lengths. After the recurrent neural network is trained, the predictions are in good agreement with the actual values. There is only marginal differences in performance of the various multi-layer perceptron configurations (confirming that the stress intensity range is relatively simple function of current crack length and far-field stress).



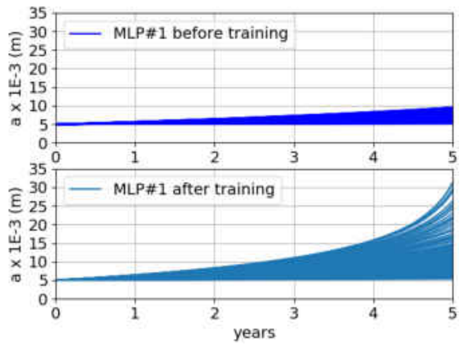
(a) Loss function history.

(b) Predictions at the 5th year for the training data (15 aircraft).

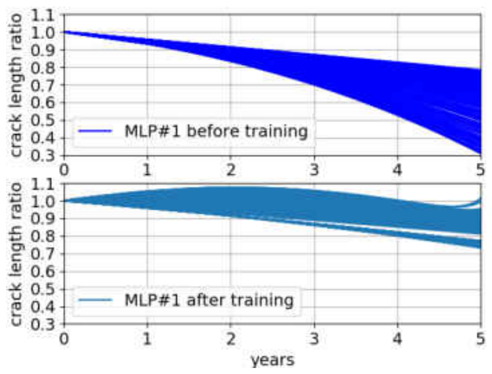
(c) Predictions at the 5th year for the entire fleet (300 aircraft).

Figure 4.12: Loss function history and predictions before and after training.

Figure 4.13 illustrates the crack length predictions over time for MLP$\#1$ (see Tab. 4.4 for details) before and after training and how they compare with the actual crack length histories. As seen in Fig. 4.13a, there is good agreement between predicted and actual crack length histories. Figure 4.13b shows the ratio between the predicted and actual crack growth over time for the entire fleet before and after training. Initially, the model grossly under-predicts large crack lengths and predictions are within a 35% and 85% of the actual crack length. After the recurrent neural network



(a) Crack length histories.

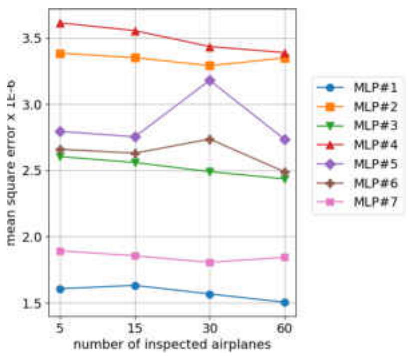


(b) Ratio between predicted and actual crack length.

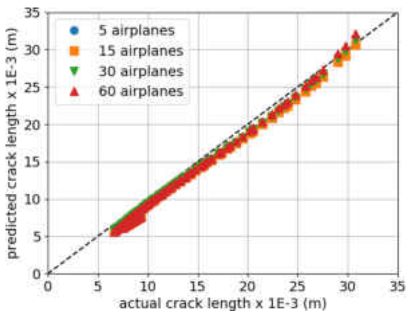
Figure 4.13: Actual and predicted crack length over time for the entire fleet (300 aircraft).

is trained, predictions stay within $\pm 15\%$ of the actual crack length, for the most part.

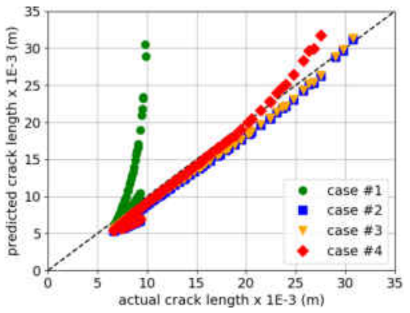
Figure 4.14 shows how the number of inspected aircraft affects the prediction accuracy of the physics-informed neural network. As mentioned before, besides time series for loads (inputs), only observations for crack at the 5th year are used for training the model. Figure 4.14a shows the mean squared error (i.e., loss function at the end of training) as a function of number of training points. Figure 4.14b shows the predictions versus actual crack lengths for the entire fleet at the end of the 5th year for MLP$\#1$. Even with as few as 5 inspected aircraft (entire load histories and crack length at the 5th year), the model is capable of producing accurate predictions. This might look counter-intuitive at first, as one would expect that the more crack length observations used for training, the more accurate the models would be. Nevertheless, we found that even for case of 5 inspected aircraft, the number of input observations (36,500) is large enough compared to the number of trainable parameters (as shown in Tab. 4.4, MLP$\#1$ has only 21 trainable parameters). On top of that, the physics-informed layer (Paris law) greatly influences the shape of the output versus time (monotonically and exponentially increasing). Therefore, the resulting physics-informed neural networks are relatively accurate even with few observed outputs.



(a) Mean squared error versus number of training points.



(b) MLP#1 predictions versus actual crack length at the 5th year for entire fleet (300 aircraft) for different number of training points.



(c) Predictions versus actual crack length at the 5th year for entire fleet (300 aircraft) when MLP#1 is trained with different data sets (see Table 4.5 and Fig 4.11 for further details).

Figure 4.14: Effect of training points in crack length predictions. Table 4.4 details the MLP configurations.

Last but not least, we also studied the effect of the distribution of crack length observations used for training the recursive neural network. For the sake of illustration, consider that the training set consists of observations for crack lengths and far-field cyclic stress at 15 different aircraft. One might be interested in looking at how well the resulting model is when the crack length observation is biased towards the low values, or towards high values, or maybe, the distribution of crack length observations does not reflect the fleet distribution. In real life, in the absence of estimators for crack length, the first planes to be inspected are chosen based on rudimentary models based on the history of flown missions. Table 4.5 and Fig 4.11 details the distribution of observed crack lengths considered here. Figure 4.14c shows the summary of results for this part of the study. Interestingly, except for case #1, the trained physics-informed neural network was able to predict crack length (there is only minor differences among cases #2, #3, and #4). This indicates that as long as the range of observed crack length covers the plausible crack length range at the fleet level, the resulting model tends to be accurate, and the distribution of observed crack length has minor effects on the resulting network.

Notes about computational cost

Our implementation is done in TensorFlow (version 2.0.0-beta1) using the Python application programming interface. In order to replicate our results, the interested reader can download codes, data, and install the PINN package (base package for physics-informed neural networks used in this work) available at [80]. In the light of the discussed application, the computational cost associated with the neural networks is considered small (training done in few minutes using a laptop computer and predictions for the 5 years of operation at a small fraction of a second per aircraft). The intended use of the models is such that after data is collected, models will be trained and prediction is performed across the fleet. The model predictions are used to decide which aircraft should be inspected next and potentially monitor the fleet with predictions done after each flight. In other words, the few minutes and fraction of second needed to run the training and prediction is negligible compared to the time between flights and between scheduled inspections (given the moderate crack growth rates for some aircraft, it is conceivable that the model is exercised between large periods of time, such as weekly runs).

CHAPTER 5: CLOSING REMARKS AND FUTURE WORK

Intended usage and limitations of the proposed framework

We believe our proposed approach can be used in several practical applications. For example, engineers and scientists could leverage physics-informed kernels that have been previously developed and proved to be able to model certain failure modes. Then, neural networks layers can be used to compensate limitations of such physics-informed kernels by modeling parts of the system that are not fully understood. This is a straightforward and practical way to characterize model-form uncertainty. Although we illustrated the case in which physics-informed and data-driven layers are connected in series, we believe the final design of the neural network architecture depends on the problem. The implementation of "MODEL" in Fig. 4.7a can combine physics-informed and data-driven layers forming complex architectures (mixing series, parallel, bridge and other arrangements).

We expect the hybrid models should perform very well in cases for which inputs are observed throughout all the time stamps but outputs are observed only at few time stamps. The physics-informed kernels are expected to compensate for the lack of output observations. In cases where both inputs and outputs are observed throughout the time stamps, we acknowledge that purely data-driven approaches could perform as well as the hybrid models.

We advocate towards the hybrid implementation, combining physics-informed and data-driven layers. As we demonstrated with the numerical experiments, we have observed that the hybrid model requires very few output observations to be trained. Unfortunately, one might also argue that the hybrid nature of the model is its main limitation. In fact, we believe there are at least two cases that can complicate the implementation of our proposed algorithm. First, it could be that the understanding of the physics is so limited that no physics-informed approximations are available. In this case, one might have to use a purely data-driven implementation (or maybe other recurrent neural network architecture, such as the long-short term memory). Even though this is a valid approach, we believe it could limit the benefits in terms of reduction in required training data. Second, the physics-informed kernels need to be fast to compute (i.e., computational cost comparable to matrix algebra found in multi-layer perceptrons). Complex models, such as those involving iterative solvers, could make the computational cost of training and prediction prohibitive, and/or make it difficult to fit the neural network.

Summary and conclusions

In this thesis, we demonstrated the ability to directly implement physics-based models into a hybrid neural network and leverage the graph-based modeling capabilities found in platforms such as Tensorflow. Specifically, our implementation inherits the capabilities offered by these frameworks such as implementation of recurrent neural network base class, automatic differentiation, and optimization methods for hyperparameter optimization.

We discussed Python implementations of ordinary differential equation solvers using recurrent neural networks with customized repeatable cells with hybrid implementation of physics-informed kernels. In our examples, we found that this approach is useful for both quantification of model-form uncertainty as well as model parameter estimation. We demonstrated our framework on two examples:

- *Euler integration of fatigue crack propagation:* our hybrid model framework characterized model form uncertainty regarding the stress intensity range used in Paris law. We implemented the numerical integration of the first order differential equation through the Euler method given that the time history of far-field stresses is available. For this case study, we observed good repeatability of results with regards to variations in initialization of the neural network hyperparameters.

- *Runge-Kutta integration of a 2 degree-of-freedom vibrations system:* our hybrid approach is capable of model parameter identification. We implemented the numerical integration of the second order differential equation through the Runge-Kutta method given that the physics is known and both inputs and outputs are observed through time. For this case study, we saw that the identified model parameters led to accurate prediction of the system displacements.

Moreover, we investigated the case of monitoring fatigue crack growth in a fleet of aircraft. We proposed a novel physics-informed recurrent neural network for cumulative damage modeling. As inputs for our cumulative damage model, we considered the current damage level (crack length) and far-field stresses (however, the recurrent neural networks can take other inputs, depending on the problem). We tested well-known recurrent neural network cells, such as the long short-term memory and the gated recurrent unit, and compare them with the proposed novel physics-informed cell for cumulative damage modeling. This proposed cell is designed such that models can be built using purely data-driven or physics-informed layers, or more interestingly, hybrids of physics-informed and data-driven layers (as the model discussed in this paper). We designed two numerical experiments in which a fleet of 300 aircraft is to be monitored. In the first scenario, on-board structural and health monitoring sensors provide crack length observations for a portion of the fleet. In the second scenario, crack length observation is obtained through scheduled inspection.

With the help of the numerical studies, we have demonstrated that recurrent neural networks can be used to model cumulative damage (here, exemplified by fatigue crack growth). For the case in which on-board structural and health monitoring sensors are installed, we learned that (a) architectures such as the long short-term memory and the gated recurrent unit tend to require frequent aircraft inspection data so that predictions of trained models can start tracking damage over time, and (b) our proposed recurrent neural network cell (hybrid of data-driven and physics-informed layers) can track damage over time even when trained with limited number of inspection data. As expected, we confirmed that the performance of purely data-driven architectures is highly dependent on the amount of data; and unfortunately, for the studied scenario, their predictions tend to be poor. For the case in which crack length data is obtained through scheduled inspection, we learned that our proposed recurrent neural network cell successfully models fatigue crack growth. We studied the effect of the distribution and number of training data in the accuracy of the crack length predictions at the fleet level after five years worth of operation. We learned that even with reduced number of data points, the proposed physics-informed neural network can approximate fatigue crack growth.

Future Work

The results motivate us to extend the study in several aspects. For example, we suggest studying the effect of improved physics of failure models (e.g., by including both initiation and propagation in the cumulative damage). We also see that considering uncertainty is important for this class of problems. For example, one can study the effect of the scatter in material properties as well as uncertainty in damage detection and quantification. Finally, one can also study how this physics-informed neural networks could be used to help decision making in operations and maintenance of industrial assets (e.g., by guiding decisions regarding repair and replacement of components).

In terms of future work, we believe there are many opportunities to advance the implementation as well as the application of the demonstrated approach. For example, one can explore further aspects of parallelization, potentially extending implementation to low-level CUDA implementation [81]. Another interesting aspect to explore would be data batching and dropout [82], which could be particularly useful when dealing with large datasets. In terms of applications, we believe that hybrid implementations like the ones we discussed are beneficial when reduced order models can capture part of the physics. Then data-driven models can compensate for the remaining uncertainty and reduce the gap between predictions and observations. In the immediate future, it would be interesting to see applications in dynamical systems and controls. For example, Altan and collaborators proposed a new model predictive controller for target tracking of a three-axis gimbal system [83] as well as a real-time control system for UAV path tracking [84]. The UAV control system is based on a nonlinear auto-regressive exogenous neural network where the proposed model predictive controller for the gimbal system is based on a Hammerstein model. The proposed control methods are used for real-time target tracking of a moving target under influence from external disturbances. It would be interesting to see how our proposed approach can be incorporated in real-time controls.

LIST OF REFERENCES

[1] Albert H. C. Tsang. Condition-based maintenance: tools and decision making. *Journal of Quality in Maintenance Engineering*, 1(3):3–17, Sep 1995.

[2] Andrew K. S. Jardine, Daming Lin, and Dragan Banjevic. A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical Systems and Signal Processing*, 20(7):1483–1510, 2006.

[3] Ying Peng, Ming Dong, and Ming Jian Zuo. Current status of machine prognostics in condition-based maintenance: a review. *The International Journal of Advanced Manufacturing Technology*, 50(1-4):297–313, Jan 2010.

[4] Suzan Alaswad and Yisha Xiang. A review on condition-based maintenance optimization models for stochastically deteriorating system. *Reliability Engineering & System Safety*, 157:54–63, 2017.

[5] Jay D. Martin, Daniel A. Finke, and Christopher B. Ligetti. On the estimation of operations and maintenance costs for defense systems. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 2478–2489, 2011.

[6] US Navy Collaborators. *Department of the Navy Total Ownership Cost (TOC) Guidebook.* Department of the Navy, 2012.

[7] GE Aviation collaborators. TrueChoice Commercial Services. Online (retrieved 14 February 2019), 2019.

[8] Siemens collaborators. Siemens Service Programs and Agreements. Online (retrieved 14 February 2019), 2019.

[9] Lufthansa Technik AG collaborators. Engine Services - Lufthansa Technik AG. Online (retrieved 14 February 2019), 2019.

[10] Gemini Energy Services collaborators. Gemini Energy Services: Wind Turbine Services. Online (retrieved 14 February 2019), 2019.

[11] Yin Cheng, Yuexin Huang, Bo Pang, and Weidong Zhang. Thermalnet: A deep reinforcement learning-based combustion optimization system for coal-fired boiler. *Engineering Applications of Artificial Intelligence*, 74:303 – 311, 2018.

[12] Changqing Shen, Yumei Qi, Jun Wang, Gaigai Cai, and Zhongkui Zhu. An automatic and robust features learning method for rotating machinery fault diagnosis based on contractive autoencoder. *Engineering Applications of Artificial Intelligence*, 76:170 – 184, 2018.

[13] Tian Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *31st Advances in Neural Information Processing Systems*, pages 6572–6583. Curran Associates, Inc., 2018.

[14] Guofei Pang and George Em Karniadakis. Physics-informed learning machines for partial differential equations: Gaussian processes versus neural networks. *Nonlinear Systems and Complexity*, pages 323–343, 2020.

[15] 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. *Journal of Computational Physics*, 378:686 – 707, 2019.

[16] Maziar Raissi and George Em Karniadakis. Hidden physics models: machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125 – 141, 2018.

[17] Shaowu Pan and Karthik Duraisamy. Physics-informed probabilistic learning of linear embeddings of nonlinear dynamics with guaranteed stability. *SIAM Journal on Applied Dynamical Systems*, 19(1):480–509, 2020.

[18] Renato G. Nascimento and Felipe A. C. Viana. Cumulative damage modeling with recurrent neural networks. *AIAA Journal*, Online First, 2020.

[19] Yigit A. Yucesan and Felipe A. C. Viana. A physics-informed neural network for wind turbine main bearing fatigue. *International Journal of Prognostics and Health Management*, 11(1):27–44, 2020.

[20] Arinan Dourado and Felipe A. C. Viana. Physics-informed neural networks for missing physics estimation in cumulative damage models: a case study in corrosion fatigue. *ASME Journal of Computing and Information Science in Engineering*, 20(6):061007 (10 pages), 2020.

[21] Anuj Karpatne, Gowtham Atluri, James H. Faghmous, Michael Steinbach, Arindam Banerjee, Auroop Ganguly, Shashi Shekhar, Nagiza Samatova, and Vipin Kumar. Theory-guided data science: A new paradigm for scientific discovery from data. *IEEE Transactions on Knowledge and Data Engineering*, 29(10):2318–2331, 2017.

[22] Shubhendu K. Singh, Ruoyu Yang, Amir Behjat, Rahul Rai, Souma Chowdhury, and Ion Matei. PI-LSTM: Physics-infused long short-term memory network. In *2019 18th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 34–41, Boca Raton, USA, Dec 2019. IEEE.

[23] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

[24] Xiao-Sheng Si, Wenbin Wang, Chang-Hua Hu, and Dong-Hua Zhou. Remaining useful life estimation – a review on the statistical data driven approaches. *European Journal of Operational Research*, 213(1):1–14, 2011.

[25] Prasanna Tamilselvan and Pingfeng Wang. Failure diagnosis using deep belief learning based health state classification. *Reliability Engineering & System Safety*, 115:124–135, 2013.

[26] Khanh Le Son, Mitra Fouladirad, Anne Barros, Eric Levrat, and Benoît Iung. Remaining useful life estimation based on stochastic deterioration models: a comparative study. *Reliability Engineering & System Safety*, 112:165–175, 2013.

[27] Gian Antonio Susto, Andrea Schirru, Simone Pampuri, Seán McLoone, and Alessandro Beghi. Machine learning for predictive maintenance: a multiple classifier approach. *IEEE Transactions on Industrial Informatics*, 11(3):812–820, 2015.

[28] Samir Khan and Takehisa Yairi. A review on the application of deep learning in system health management. *Mechanical Systems and Signal Processing*, 107:241–265, 2018.

[29] Thilo Stadelmann, Vasily Tolkachev, Beate Sick, Jan Stampfli, and Oliver Dürr. *Applied Data Science-Lessons Learned for the Data-Driven Business*, chapter Beyond ImageNet-deep learning in industrial practice, pages 205–232. Springer, 2018.

[30] Matthew J. Daigle and Kai Goebel. Model-based prognostics with concurrent damage progression processes. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(3):535–546, 2013.

[31] Chenzhao Li, Sankaran Mahadevan, You Ling, Sergio Choze, and Liping Wang. Dynamic Bayesian network for aircraft wing health monitoring digital twin. *AIAA Journal*, 55(3):930–941, 2017.

[32] You Ling, Isaac Asher, Liping Wang, Felipe A. C. Viana, and Genghis Khan. Information gain-based inspection scheduling for fatigued aircraft components. In *AIAA Scitech 2017 Forum*, pages AIAA–2017–1565, Grapevine, USA, Jan 2017. AIAA.

[33] Yigit A. Yucesan and Felipe A. C. Viana. Onshore wind turbine main bearing reliability and its implications in fleet management. In *AIAA Scitech 2019 Forum*, pages AIAA–2019–1225, Orlando, USA, Jan 2019. AIAA.

[34] Pier Carlo C. Berri, Matteo Davide Lorenzo Dalla Vedova, and Laura Mainini. Real-time fault detection and prognostics for aircraft actuation systems. In *AIAA Scitech 2019 Forum*, pages AIAA–2019–2210, San Diego, USA, Jan 2019. AIAA.

[35] Carl S Byington, Matthew Watson, and Doug Edwards. Data-driven neural network methodology to remaining life predictions for aircraft actuator components. In *2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No. 04TH8720)*, volume 6, pages 3581–3589. IEEE, 2004.

[36] Andrea De Martin Giovanni Jacazio and Massimo Sorli. Enhanced particle filter framework for improved prognosis of electro-mechanical flight controls actuators. In *Fourth European Conference of the Prognosis and Health Management Society*, page 10 pages, Utrecht, The Netherlands, 2018. PHM Society.

[37] Hyuk Lee and In Seok Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110–131, 1990.

[38] Andrew J. Meade Jr. and Alvaro A. Fernandez. Solution of nonlinear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, 20(9):19–44, 1994.

[39] Jun Takeuchi and Yukio Kosugi. Neural network representation of finite element method. *Neural Networks*, 7(2):389–395, 1994.

[40] Yi-Jen Wang and Chin-Teng Lin. Runge-kutta neural network for identification of dynamical systems in high accuracy. *IEEE Transactions on Neural Networks*, 9(2):294–307, 1998.

[41] Shankar Sankararaman, Kyle McLemore, Sankaran Mahadevan, Samuel C. Bradford, and Lee D. Peterson. Test resource allocation in hierarchical systems using Bayesian networks. *AIAA Journal*, 51(3):537–550, 2013.

[42] Felipe A. C. Viana, Timothy W. Simpson, Vladimir Balabanov, and Vassili Toropov. Metamodeling in multidisciplinary design optimization: how far have we really come? *AIAA Journal*, 52(4):670–690, 2014.

[43] Hongyi Xu, Ruoqian Liu, Alok Choudhary, and Wei Chen. A machine learning-based design representation method for designing heterogeneous microstructures. *Journal of Mechanical Design*, 137(5):051403, 2015.

[44] Anand Pratap Singh, Shivaji Medida, and Karthik Duraisamy. Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils. *AIAA Journal*, 55(7):2215–2227, 2017.

[45] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Numerical Gaussian processes for time-dependent and nonlinear partial differential equations. *SIAM Journal on Scientific Computing*, 40(1):A172–A198, 2018.

[46] J. Nagoor Kani and Ahmed H. Elsheikh. Reduced-order modeling of subsurface multi-phase flow models using deep residual recurrent neural networks. *Transport in Porous Media*, 126(3):713–741, 2018.

[47] Maziar Raissi. Deep hidden physics models: deep learning of nonlinear partial differential equations. *Journal of Machine Learning Research*, 19(25):1–24, 2018.

[48] Jin-Long Wu, Heng Xiao, and Eric Paterson. Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework. *Physical Review Fluids*, 3(7):074602, 2018.

[49] Jan S. Hesthaven and Stefano Ubbiali. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics*, 363:55–78, 2018.

[50] Renee Swischuk, Laura Mainini, Benjamin Peherstorfer, and Karen Willcox. Projection-based model reduction: Formulations for physics-based machine learning. *Computers & Fluids*, 179:704–717, 2019.

[51] Michael Schober, David K. Duvenaud, and Philipp Hennig. Probabilistic ODE solvers with Runge-Kutta means. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *27th Advances in Neural Information Processing Systems*, pages 739–747. Curran Associates, Inc., 2014.

[52] J. Nagoor Kani and Ahmed H. Elsheikh. DR-RNN: A deep residual recurrent neural network for model reduction. *arXiv preprint arXiv:1709.00939*, 2017.

[53] David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[54] Kevin P. Murphy. *Machine learning: a probabilistic perspective*. MIT Press, 2012.

[55] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[56] Barak A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989.

[57] Alex Aussem. Dynamical recurrent neural networks towards prediction and modeling of dynamical systems. *Neurocomputing*, 28(1-3):207–232, 1999.

[58] Sucheta Chauhan and Lovekesh Vig. Anomaly detection in ECG time signals via deep long short-term memory networks. In *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–7, Oct 2015.

[59] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

[60] Mei Yuan, Yuting Wu, and Li Lin. Fault diagnosis and remaining useful life estimation of aero engine using lstm neural network. In *2016 IEEE International Conference on Aircraft Utility Systems (AUS)*, pages 135–140. IEEE, 2016.

[61] Liang Guo, Naipeng Li, Feng Jia, Yaguo Lei, and Jing Lin. A recurrent neural network based health indicator for remaining useful life prediction of bearings. *Neurocomputing*, 240:98–109, 2017.

[62] Gae-Won You, Sangdo Park, and Dukjin Oh. Diagnosis of electric vehicle batteries using recurrent neural networks. *IEEE Transactions on Industrial Electronics*, 64(6):4885–4893, 2017.

[63] Qianhui Wu, Keqin Ding, and Biqing Huang. Approach for fault prognosis using recurrent neural network. *Journal of Intelligent Manufacturing*, pages 1–13, 2018.

[64] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[65] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In Francis Bach and David Blei, editors, *Proceedings of the 32nd In-*

ternational Conference on Machine Learning, volume 37 of *Proceedings of Machine Learning Research*, pages 2067–2075, Lille, France, 07–09 Jul 2015. PMLR.

[66] Pe Paris and Fazil Erdogan. A critical analysis of crack propagation laws. *Journal of Basic Engineering*, 85(4):528–533, 1963.

[67] Norman E. Dowling. *Mechanical Behavior of Materials: Engineering Methods for Deformation, Fracture, and Fatigue*. Pearson, 2012.

[68] Jack A. Collins. *Failure of Materials in Mechanical Design: Analysis, Prediction, Prevention*. John Wiley & Sons, 1993.

[69] Kajetan Fricke, Renato G. Nascimento, and Felipe A. C. Viana. Python implementation of ordinary differential equations solvers using hybrid physics-informed neural networks. https://github.com/PML-UCF/pinn_ode_tutorial, May 2020.

[70] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, New York, USA, September 2007.

[71] J.C. Butcher and G. Wanner. Runge-kutta methods: some historical notes. *Applied Numerical Mathematics*, 22(1):113 – 151, 1996. Special Issue Celebrating the Centenary of Runge-Kutta Methods.

[72] D. Roach. Real time crack detection using mountable comparative vacuum monitoring sensors. *Smart Structures and Systems*, 5(4):317–328, 2009.

[73] Kenneth O. Hill and Gerald Meltz. Fiber bragg grating technology fundamentals and overview. *Journal of Lightwave Technology*, 15(8):1263–1276, 1997.

[74] John C. Aldrin, Jeremy S. Knopp, Eric A. Lindgren, and Kumar V. Jata. Model-assisted probability of detection evaluation for eddy current inspection of fastener sites. In *AIP Conference Proceedings*, volume 1096, pages 1784–1791. AIP, 2009.

[75] B. W. Drinkwater and P. D. Wilcox. Ultrasonic arrays for non-destructive evaluation: A review. *NDT & E International*, 39(7):525 – 541, 2006.

[76] Michael Van Hoye. Fluorescent penetrant crack detection. Patent: US4621193A, Nov 1986.

[77] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, USA, 17–19 Jun 2013. PMLR.

[78] Ilya Sutskever. *Training recurrent neural networks*. PhD thesis, University of Toronto Toronto, Ontario, Canada, 2013.

[79] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arxiv 2014. *arXiv preprint arXiv:1412.6980*, 2014.

[80] Felipe A. C. Viana, Renato G. Nascimento, Yigit Yucesan, and Arinan Dourado. Physics-informed neural networks package. https://github.com/PML-UCF/pinn, Aug 2019.

[81] TensorFlow Contributors. Create an op. https://www.tensorflow.org/guide/create_op, Aug 2020.

[82] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[83] A. Altan and Rifat Hacioglu. Model predictive control of three-axis gimbal system mounted on UAV for real-time target tracking under external disturbances. *Mechanical Systems and Signal Processing*, 138, 2020.

[84] A. Altan, O. Aslan, and Rifat Hacioglu. Real-time control based NARX neural networks of hexarotor UAV with load transporting system for path tracking. In *2018 6th International Conference on Control Engineering Information Technology (CEIT)*, pages 1–6, Istanbul, Turkey, 2018. IEEE.