Machine Learning: Science and Technology

PAPER • OPEN ACCESS

# Fractional deep neural network via constrained optimization

To cite this article: Harbir Antil *et al* 2021 *Mach. Learn.: Sci. Technol.* **2** 015003

View the article online for updates and enhancements.

This content was downloaded from IP address 106.213.19.213 on 30/06/2023 at 08:58

MACHINE LEARNING Science and Technology

CrossMark

OPEN ACCESS

RECEIVED
23 April 2020

REVISED
13 July 2020

ACCEPTED FOR PUBLICATION
23 July 2020

PUBLISHED
1 December 2020

Original Content from this work may be used under the terms of the Creative Commons Attribution 4.0 licence.

Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

PAPER

# Fractional deep neural network via constrained optimization

**Harbir Antil**[1], **Ratna Khatri**[1], **Rainald Löhner**[2] **and Deepanshu Verma**[1]

[1] Department of Mathematical Sciences and the Center for Mathematics and Artificial Intelligence (CMAI), George Mason University, Fairfax, VA 22030, United States of America
[2] Center for Computational Fluid Dynamics, George Mason University, Fairfax, VA, United States of America

**E-mail:** hantil@gmu.edu, rkhatri3@gmu.edu, dverma2@gmu.edu, rlohner@gmu.edu

**Keywords:** deep learning, deep neural network, fractional time derivatives, constrained optimization, fractional neural network

## Abstract

This paper introduces a novel algorithmic framework for a deep neural network (DNN), which in a mathematically rigorous manner, allows us to incorporate history (or memory) into the network—it ensures all layers are connected to one another. This DNN, called Fractional-DNN, can be viewed as a time-discretization of a fractional in time non-linear ordinary differential equation (ODE). The learning problem then is a minimization problem subject to that fractional ODE as constraints. We emphasize that an analogy between the existing DNN and ODEs, with standard time derivative, is well-known by now. The focus of our work is the Fractional-DNN. Using the Lagrangian approach, we provide a derivation of the backward propagation and the design equations. We test our network on several datasets for classification problems. Fractional-DNN offers various advantages over the existing DNN. The key benefits are a significant improvement to the vanishing gradient issue due to the memory effect, and better handling of nonsmooth data due to the network's ability to approximate non-smooth functions.

## 1. Introduction

Deep learning has emerged as a potent area of research and has enabled a remarkable progress in recent years spanning domains like imaging science [1–4], biomedical applications [5–7], satellite imagery, remote sensing [8–10], etc. However, the mathematical foundations of many machine learning architectures are largely lacking [11–15]. The current trend of success is largely due to the empirical evidence. Due to the lack of mathematical foundation, it becomes challenging to understand the detailed workings of networks [16, 17].

The overarching goal of machine learning algorithms is to learn a function using some known data. Deep Neural Networks (DNN), like Residual Neural Networks (RNN), are a popular family of deep learning architectures which have turned out to be groundbreaking in imaging science. An introductory example of RNN is the ResNet [1] which has been successful for classification problems in imaging science. Compared to the classical DNNs, the innovation of the RNN architecture comes from a simple addition of an identity map between each layer of the network. This ensures a continued flow of information from one layer to another. Despite their success, DNNs are prone to various challenges such as vanishing gradients [11, 18, 19], difficulty in approximating non-smooth functions, long training time [20], etc.

We remark that recently in [21] the authors have introduced a DenseNet, which is a new approach to prevent the gradient 'wash out' by considering dense blocks, in which each layer takes into account all the previous layers (or the memory). They proceed by concatenating the outputs of each dense block which is then fed as an input to the next dense block. Clearly as the number of layers grow, it can become prohibitively expensive for information to propagate through the network. DenseNet can potentially overcome the vanishing gradient issue, but it is only an adhoc method [21, 22]. Some other networks that have attempted to induce multilayer connections are Highway Net [23], AdaNet [24], ResNetPlus [25], etc. Furthermore, rigorous approaches to learn nonsmooth functions such as the absolute value function $|x|$ are scarce [26].

There has been a recent push in the scientific community to develop rigorous mathematical models and understanding of the DNNs [15]. One way of doing so is to look at their architecture as dynamical systems.

© 2020 The Author(s). Published by IOP Publishing Ltd

The articles [14, 27–30] have established that a DNN can be regarded as an optimization problem subject to a discrete ordinary differential equation (ODE) as constraints. The limiting problem in the continuous setting is an ODE constrained optimization problem [14, 29]. Notice that designing the solution algorithms at the continuous level can lead to architecture independence, i.e. the number of iterations remains the same even if the number of layers is increased.

The purpose of this paper is to present a novel fractional deep neural network which allows the network to access historic information of input and gradients across all subsequent layers. This is facilitated via our proposed use of fractional derivative based ODE as constraints. We derive the optimality conditions for this network using the Lagrangian approach. Next, we consider a discretization for this fractional ODE and the resulting DNN is called *Fractional-DNN*. We provide the algorithm and show numerical examples on some standard datasets.

For completeness, we also mention the Fractional Physics Informed Neural Networks (fPINNs) [31], where the authors aim to solve Partial Differential Equations (PDEs), in particular fractional PDEs. This is an extension of authors' previous works in [32, 33]. Their idea is to minimize the PDE residual in a least-squares formulation and learn the unknown parameters using a standard feedforward NN. We also mention [34, 35], where authors solve fractional differential equations using artificial neural networks. This is completely different than what we propose in this paper. Our goal is to introduce a new NN with memory using the fractional derivatives. We formulate the NN problem as an ODE constrained optimal control problem and use Lagrangian formulation to derive the optimality conditions. We apply our NN to classification problems but notice that it can also be applied to the problems discussed in [31, 32].

Fractional derivatives appear in numerous scientific and engineering domains. The application of fractional derivatives onto functions requires less smoothness on these functions, in comparison with the integer order derivatives. The main idea behind fractional derivatives is to replace the 'derivatives' by 'integrals'. The resulting fractional operators are nonlocal, which naturally enables long range interactions. This idea is being widely used, for example, in Peridynamics which has recently emerged as a new branch of mechanics. Peridynamics can naturally capture nonsmooth effects such as crack formation [36, 37]. Notice that the classical derivatives do not make sense in that nonsmooth setting. As in the classical setting, various stochastic models can also describe the occurrence of fractional derivatives in diffusion processes. For example, fractional Laplacian corresponds to long jump random walk. If the particle experiences delays between the jumps, one arrives at the fractional time derivatives, see [38–41] and the references therein. By using the first principle arguments, the fractional Laplacian has also made an appearance, recently, in geophysics [42, 43]. Fractional Laplacian is also being used extensively as a regularizer in inverse problems [2, 44, 45]. On the other hand, fractional time derivative can capture hereditary or memory effects in materials [46].

The inherent nonlocality of fractional operators and their flexibility in applications to nonsmooth functions, is the key motivation for us to use them in Deep Learning. In the architectural setting of Deep Learning algorithms, this nonlocality translates to long-range connections between the network layers, i.e. it goes beyond the simple addition of skip connections. In [47], the authors have considered a dynamical system representation of a classical RNN, which makes use of integer order derivative to model single layer skip connection. In our work, we build upon the same model by replacing integer order time derivative with fractional order time derivative. In particular, fractional time derivative allows memory effects, therefore, in our proposed Fractional-DNN all the layers are connected to one another. In addition, fractional time derivatives can be applied to nonsmooth functions [48]. Fractional calculus has also been used in optimal control problem, see [49–51]. Thus, we aim to keep the benefits of standard DNN and the ideology of DenseNet, but remove the bottlenecks.

There are several benefits of using fractional order derivatives in machine learning architectures. First, they facilitate modeling of multi-layer connections. This in turn leads to an improved handling of the vanishing gradient issue, persistent in deep learning architectures. We have demonstrated this numerically, in this paper, through our proposed Fractional-DNN algorithm. Secondly, they allow approximation of functions, i.e. network architectures, which are not smooth. We have demonstrated how fractional time derivatives can be applied to nonsmooth functions. Thirdly, we have empirically observed that our proposed Fractional-DNN brings about an improvement in the learning rate, which is an important hyper-parameter that influences training [52]. Lastly, use of fractional order derivatives in deep learning brings the two fields of optimal control and deep learning closer, where both fields can benefit from the tools that have been independently discovered thus forth, see for instance [2, 30, 31, 47, 53].

The paper is organized as follows. In section 2 we introduce notations and definitions. We introduce our proposed Fractional-DNN in section 3. This is followed by section 4 where we discuss its numerical approximation. In section 5, we state our algorithm. The numerical examples given in section 6 show the working and improvements due to the proposed ideas on three different datasets.

**Table 1.** Table of Notations.

| Symbol | Description |
|---|---|
| $n \in \mathbb{N}$ | Number of distinct samples |
| $n_f \in \mathbb{N}$ | Number of sample features |
| $n_c \in \mathbb{N}$ | Number of classes |
| $N \in \mathbb{N}$ | Number of network layers (i.e. network depth) |
| $Y \in \mathbb{R}^{n_f \times n}$ | $Y = \{y^{(i)}\}_{i=1}^{n}$ is the collective feature set of $n$ samples. |
| $C_{obs} \in \mathbb{R}^{n_c \times n}$ | $C_{obs} = \{c^{(i)}\}_{i=1}^{n}$ are the true class labels of the input data |
| $W \in \mathbb{R}^{n_c \times n_f}$ | Weights |
| $K \in \mathbb{R}^{n_f \times n_f}$ | Linear operator (distinct for each layer) |
| $b \in \mathbb{R}$ | Bias (distinct for each layer) |
| $P \in \mathbb{R}^{n_f \times n}$ | Lagrange multiplier |
| $e_{n_c} \in \mathbb{R}^{n_c}$ | A vector of ones |
| $\tau \in \mathbb{R}$ | Time step-length |
| $\sigma(\cdot)$ | Activation function, acting pointwise |
| $\gamma$ | Order of fractional time derivative |
| $(\cdot)'$ | Derivative w.r.t. the argument |
| $tr(\cdot)$ | Trace operator |
| $(\cdot)^\intercal$ | Matrix transpose |
| $\odot$ | Point-wise multiplication |
| $m_1$ | Max count for randomly selecting a mini-batch in training |
| $m_2$ | Max iteration count for gradient-based optimization solver |
| $\alpha_{train},\ \alpha_{test}$ | Percentage of training and testing data correctly identified |

## 2. Preliminaries

The purpose of this section is to introduce some notations and definitions that we will use throughout the paper. We begin with table 1 where we state the standard notations. In section 2.1 we describe the well-known softmax loss function. Section 2.2 is dedicated to the Caputo fractional time derivative.

### 2.1. Cross entropy with softmax function

Given collective feature matrix $Y$ with true labels $C_{obs}$ and the unknown weights $W$, the cross entropy loss function, given by

$$E(W, Y, C_{obs}) = -\frac{1}{n}\,\mathrm{tr}(C_{obs}^\intercal \log(S(W, Y))), \tag{1}$$

measures the discrepancy between the true labels $C_{obs}$ and the predicted labels $\log(S(W, Y))$. Here,

$$S(W, Y) := \exp(WY)\,\mathrm{diag}\left(\frac{1}{e_{n_c}^\intercal \exp(WY)}\right) \tag{2}$$

is the softmax classifier function, which gives normalized probabilities of samples belonging to the classes.

### 2.2. Caputo fractional derivative

In this section, we define the notion of Caputo fractional derivative and refer [54, equations (2.4.17) and (2.4.18)], also see [55], for the following definitions.

**Definition 2.1** (Left Caputo Fractional Derivative). *For a fixed real number $0 < \gamma < 1$, and an absolutely continuous function $u\colon [0, T] \to \mathbb{R}$, the left Caputo fractional derivative is defined by:*

$$d_t^\gamma u(t) = \frac{1}{\Gamma(1-\gamma)} \int_0^t \frac{u'(r)}{(t-r)^\gamma}\, dr, \tag{3}$$

*where $\Gamma(\cdot)$ is the Euler-Gamma function.*

**Definition 2.2** (Right Caputo Fractional Derivative). *For a fixed real number $0 < \gamma < 1$, and an absolutely continuous function $u\colon [0, T] \to \mathbb{R}$, the right Caputo fractional derivative is defined by*:

$$d_{T-t}^\gamma u(t) = \frac{-1}{\Gamma(1-\gamma)} \int_t^T \frac{u'(r)}{(r-t)^\gamma}\, dr. \tag{4}$$

Moreover, if $\gamma = 1$ and $u \in C^1([0,T])$, then one can show that $d_t^\gamma u(t) = u'(t) = d_{T-t}^\gamma u(t)$. We note that the fractional derivatives in equation (3) and equation (4) are nonlocal operators. Indeed, the derivative of $u$ at a point $t$ depends on all the past and future events, respectively. This behavior is different than the classical case of $\gamma = 1$.

The left and right Caputo fractional derivatives are linked by the following fractional integration by parts formula, [49, Lemma 3], for $\gamma \in (0,1)$, let

$$\mathbb{L}_\gamma := \left\{ f \in C([0,T]) : d_t^\gamma f \in L^2(0,T) \right\}, \quad \mathbb{R}_\gamma := \left\{ f \in C([0,T]) : d_{T-t}^\gamma f \in L^2(0,T) \right\}.$$

**Lemma 2.3** (Fractional Integration-by-Parts). *For $f \in \mathbb{L}_\gamma$ and $g \in \mathbb{R}_\gamma$, the following integration-by-parts formula holds:*

$$\int_0^T d_t^\gamma f(t) g(t)\, dt = \int_0^T f(t) d_{T-t}^\gamma g(t)\, dt + g(T)(I_t^{1-\gamma} f)(T) - f(0)(I_{T-t}^{1-\gamma} g)(0), \tag{5}$$

*where $I_t^{1-\gamma} w(t)$ and $I_{T-t}^{1-\gamma} w(t)$ are the left and right Riemann-Liouville fractional integrals of order $\gamma$ and are given by*

$$I_t^{1-\gamma} w(t) := \frac{1}{\Gamma(1-\gamma)} \int_0^t \frac{w(r)}{(t-r)^\gamma}\, dr \text{ and } I_{T-t}^{1-\gamma} w(t) := \frac{1}{\Gamma(1-\gamma)} \int_t^T \frac{w(r)}{(r-t)^\gamma}\, dr.$$

## 3. Continuous fractional deep neural network

After the above preparations, in this section, we shall introduce the Fractional-DNN. First we briefly describe the classical RNN, and then extend it to develop the Fractional-DNN. We formulate our problem as a constrained optimization problem. Subsequently, we shall use the Lagrangian approach to derive the optimality conditions.

### 3.1. Classical RNN

Our goal is to approximate a map $\mathcal{F}$. A classical RNN helps approximate $\mathcal{F}$, for a known set of inputs and outputs. To construct an RNN, for each layer $j$, we first consider a linear-transformation of $Y_{j-1}$ as,

$$\mathcal{G}_{j-1}(Y_{j-1}) = K_{j-1} Y_{j-1} + b_{j-1},$$

where the pair $(K_j, b_j)$ denotes an unknown linear operator and bias at the $j^{th}$ layer. When $N > 1$, the network is considered 'deep'. Next we introduce non-linearity using a non-linear activation function $\sigma$ (e.g. ReLU or tanh). The resulting RNN is,

$$Y_j = Y_{j-1} + \tau(\sigma \circ \mathcal{G}_{j-1})(Y_{j-1}), \quad j = 1, \cdots, N; \quad N > 1, \tag{6}$$

where $\tau > 0$ is the time-step. Finally, the RNN approximation of $\mathcal{F}$ is given by,

$$\mathcal{F}_\theta(\cdot) = \Big( \big(I + \tau(\sigma \circ \mathcal{G}_{N-1})\big) \circ \big(I + \tau(\sigma \circ \mathcal{G}_{N-2})\big) \circ \cdots \circ \big(I + \tau(\sigma \circ \mathcal{G}_0)\big) \Big)(\cdot),$$

with $\theta = (K_j, b_j)$ as the unknown parameters. In other words, the problem of approximating $\mathcal{F}$ using classical RNN, intrinsically, is a problem of learning $(K_j, b_j)$.

Hence, for given datum $(Y_0, C)$, the learning problem then reduces to minimizing a loss function $\mathcal{J}(\theta, (Y_N, C))$, subject to constraint equation (6), i.e.

$$\begin{aligned} &\min_\theta \quad \mathcal{J}(\theta, (Y_N, C)) \\ &\text{s.t.} \quad Y_j = Y_{j-1} + \tau(\sigma \circ \mathcal{G}_{j-1})(Y_{j-1}), \quad j = 1, \ldots, N. \end{aligned} \tag{7}$$

Notice that the system equation (6) is the forward-Euler discretization of the following continuous in time ODE, see [1, 14, 47],

$$\begin{aligned} d_t Y(t) &= \sigma(K(t) Y(t) + b(t)), \quad t \in (0,T), \\ Y(0) &= Y_0. \end{aligned} \tag{8}$$

The continuous learning problem then requires minimizing the loss function $\mathcal{J}$ at the final time $T$ subject to the ODE constraints equation (8):

$$\begin{aligned} &\min_{\theta=(K,b)} \quad \mathcal{J}(\theta,(Y(T),C)) \\ &\text{s.t.} \quad d_t Y(t) = \sigma(K(t)Y(t)+b(t)), \quad t\in(0,T), \\ &\qquad Y(0)=Y_0. \end{aligned} \tag{9}$$

Notice that designing algorithms for solving the continuous in time problem equation (9) instead of the discrete in time problem equation (7) has several key advantages. In particular, it will lead to algorithms which are independent of the neural network architecture, i.e. independent of the number of layers. In addition, the approach used in solving problem equation (9) can help us determine the stability of the neural network equation (7), see [27, 30]. Moreover, for the neural network equation (7), it has been noted that as the information about the input or gradient passes through many layers, it can vanish and 'wash out', or grow and 'explode' exponentially [18]. There have been adhoc attempts to address these concerns, see for instance [21, 23, 24], but a satisfactory mathematical explanation and model does not currently exist. One of the main goals of this paper is to introduce such a model.

Notice that equation (8), and its discrete version equation (6), incorporates many algorithmic processes such as linear solvers, preconditioners, non-linear solvers, optimization solvers, etc. Furthermore, there are well-established numerical algorithms that re-use information from previous iterations to accelerate convergence, e.g. the BFGS method [56], Anderson acceleration [57], and variance reduction methods [58]. These methods account for the history $Y_j, Y_{j-1}, Y_{j-2}, \ldots, Y_0$, while choosing $Y_{j+1}$. Motivated by these observations we introduce versions of equation (6) and equation (8) that can account for history (or memory) effects in a rigorous mathematical fashion.

### 3.2. Continuous fractional-DNN

The fractional time derivative given by equation (3) has a distinct ability to allow a memory effect, for instance in materials with hereditary properties [46]. Fractional time derivative can be derived by using the anomalous random walks where the walker experiences delays between jumps [38]. In contrast, the standard time derivative naturally arises in the case of classical random walks. We use the idea of fractional time derivative to enrich the constraint optimization problem equation (9), and subsequently equation (7), by replacing the standard time derivative $d_t$ by the fractional time derivative $d_t^\gamma$ of order $\gamma\in(0,1)$. Recall that for $\gamma=1$, we obtain the classical derivative $d_t$. Our new continuous in time model, the Fractional-DNN, is then given by (cf equation (8)),

$$\begin{aligned} &d_t^\gamma Y(t) = \mathcal{F}_\theta(Y(t),t,\theta(t)), \quad t\in(0,T), \\ &Y(0)=Y_0, \end{aligned} \tag{10}$$

where $d_t^\gamma$ is the Caputo fractional derivative as defined in equation (3). The discrete formulation of Fractional-DNN will be discussed in the subsequent section.

The main reason for using the Caputo fractional time derivative over its other counterparts such as the Riemann Liouville fractional derivative is the fact that the Caputo derivative of a constant function is zero and one can impose the initial conditions $Y(0)=Y_0$ in a classical manner [59]. Note that $d_t^\gamma$ is a nonlocal operator in a sense that in order to evaluate the fractional derivative of $Y$ at a point $t$, we need the cumulative information of $Y$ over the entire sub-interval $[0,t)$. This is how the Fractional-DNN enables connectivity across all antecedent layers (hence the memory effect). As we shall illustrate with the help of a numerical example in section 6, this feature can help overcome the vanishing gradient issue, as the cumulative effect of the gradient of the precedent layers is less likely to be zero.

The generic learning problem with Fractional-DNN as constraints can be expressed as,

$$\begin{aligned} &\min_{\theta=(K,b)} \quad \mathcal{J}(\theta,(Y(T),C)) \\ &\text{s.t.} \quad d_t^\gamma Y(t) = \mathcal{F}_\theta(Y(t),t,\theta(t)), \quad t\in(0,T), \\ &\qquad Y(0)=Y_0. \end{aligned} \tag{11}$$

Note that the choice of $\mathcal{J}$ depends on the type of learning problem. We will next consider a specific structure of $\mathcal{J}$ given by the cross entropy loss functional equation (1).

### 3.3. Continuous fractional-DNN and cross entropy loss functional

Supervised learning problems are a broad class of machine learning problems which use labeled data. These problems are further divided into two types, namely regression problems and classification problems. The specific type of the problem dictates the choice of $\mathcal{J}$ in equation (11). Regression problems often occur in physics informed models, e.g. sample reconstruction inverse problems [2, 7]. On the other hand, classification problems occur, for instance, in computer vision [60, 61]. In both the cases, a neural network is used to learn the unknown parameters. In the discussion below we shall focus on classification problems, however, the entire discussion directly applies to regression type problems, as well.

Recall that the cross entropy loss functional $E$, defined in equation (1), measures the discrepancy between the actual and the predicted classes. Replacing, $\mathcal{J}$ in equation (11) by $E$ together with a regularization term $\mathcal{R}(W, K(t), b(t))$, we arrive at

$$\begin{aligned} &\min_{W,K,b} E(W, Y(T), C_{obs}) + \mathcal{R}(W, K(t), b(t)) \\ &\text{s.t.} \begin{cases} d_t^\gamma Y(t) = \sigma(K(t)Y(t) + b(t)), & t \in (0, T), \\ Y(0) = Y_0. \end{cases} \end{aligned} \tag{12}$$

Note that, in this case, the unknown parameter $\theta := (W, K, b)$, where $K$ and $b$ are, respectively, the linear operator and bias for each layer, and the weights $W$ are a feature-to-class map. Furthermore, $\sigma$ is a non-linear activation function and $(Y_0, C_{obs})$ is the given data, with $C_{obs}$ as the true labels of $Y_0$.

To solve equation (12), we rewrite this problem as an unconstrained optimization problem via the Lagrangian functional and derive the optimality conditions. Let $P$ denote the Lagrange multiplier, then the Lagrangian functional is given by,

$$\mathcal{L}(Y, W, K, b; P) := E(W, Y(T), C_{obs}) + \mathcal{R}(W, K(t), b(t)) + \langle d_t^\gamma Y(t) - \sigma(K(t)Y(t) + b(t)), P(t)\rangle,$$

where, $\langle \cdot, \cdot \rangle := \int_0^T \langle \cdot, \cdot \rangle_F \, dt$ is the $L^2$-inner product, and $\langle \cdot, \cdot \rangle_F$ is the Frobenius inner product. Using the fractional integration-by-parts formula equation (5), we obtain

$$\begin{aligned} \mathcal{L}(Y, W, K, b; P) = E(W, Y(T), C_{obs}) &+ \mathcal{R}(W, K(t), b(t)) - \langle \sigma(K(t)Y(t) + b(t)), P(t)\rangle \\ &+ \langle Y(t), d_{T-t}^\gamma P(t)\rangle + \langle (I_t^{1-\gamma} Y)(T), P(T)\rangle_F - \langle Y_0, (I_{T-t}^{1-\gamma} P)(0)\rangle_F. \end{aligned} \tag{13}$$

Let $(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P})$ denote a stationary point, then the first order necessary optimality conditions are given by the following state, adjoint and design equations:

(a) **State equation.** The gradient of $\mathcal{L}$ with respect to $P$ at $(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P})$ yields the state equation $\nabla_P \mathcal{L}(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P}) = 0$, equivalently,

$$\begin{aligned} d_t^\gamma \overline{Y}(t) &= \sigma(\overline{K}(t)\overline{Y}(t) + \overline{b}(t)), \qquad t \in (0, T), \\ \overline{Y}(0) &= Y_0, \end{aligned} \tag{14}$$

where $d_t^\gamma$ denotes the left Caputo fractional derivative given by equation (3). For the state variable $\overline{Y}$, we solve equation (14) forward in time, therefore we call equation (14) as the *forward propagation*.

(b) **Adjoint equation.** Next, the gradient of $\mathcal{L}$ with respect to $Y$ at $(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P})$ yields the adjoint equation $\nabla_Y \mathcal{L}(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P}) = 0$, equivalently,

$$\begin{aligned} d_{T-t}^\gamma \overline{P}(t) &= (\sigma'(\overline{K}(t)\overline{Y}(t) + \overline{b}(t))\, \overline{K}(t))^\intercal\, \overline{P}(t) \\ &= \overline{K}(t)^\intercal \left(\overline{P}(t) \odot \sigma'\left(\overline{K}(t)\overline{Y}(t) + \overline{b}(t)\right)\right), \qquad t \in (0, T), \\ \overline{P}(T) &= -\frac{1}{n}\overline{W}^\intercal(-C_{obs} + S(\overline{W}, \overline{Y}(T))), \end{aligned} \tag{15}$$

where $d_{T-t}^\gamma$ denotes the right Caputo fractional derivative given by equation (4) and $S$ is the softmax function defined in equation (2). Notice that the adjoint variable $\overline{P}$ in equation (15), with its terminal condition, is obtained by marching backward in time. As a result, the equation (15) is called *backward propagation*.

(c) **Design equations.** Finally, equating $\nabla_W \mathcal{L}(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P})$, $\nabla_K \mathcal{L}(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P})$, and $\nabla_b \mathcal{L}(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P})$ to zero, respectively, yields the design equations (with $(\overline{W}, \overline{K}, \overline{b})$ as the design variables),

$$\begin{aligned}\nabla_W \mathcal{L}(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P}) &= \frac{1}{n}\left(-C_{obs} + S(\overline{W}, \overline{Y}(T))\right)\left(\overline{Y}(T)\right)^{\intercal} \\ &\quad + \nabla_W \mathcal{R}(\overline{W}, \overline{K}(T), \overline{b}(T)) = 0, \\ \nabla_K \mathcal{L}(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P}) &= -\overline{Y}(t)\left(\overline{P}(t) \odot \sigma'(\overline{K}(t)\overline{Y}(t) + \overline{b}(t))\right)^{\intercal} \\ &\quad + \nabla_K \mathcal{R}(\overline{W}, \overline{K}(t), \overline{b}(t)) = 0, \\ \nabla_b \mathcal{L}(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P}) &= -\langle \sigma'(\overline{K}(t)\overline{Y}(t) + \overline{b}(t)), \overline{P}(t)\rangle_F \\ &\quad + \nabla_b \mathcal{R}(\overline{W}, \overline{K}(t), \overline{b}(t)) = 0,\end{aligned} \tag{16}$$

for almost every $t \in (0, T)$.

In view of (a)–(c), we can use a gradient based solver to find a stationary point to problem equation (12).

**Remark 3.1** (Parametric Kernel $K(\psi(t))$). *Throughout our discussion, we have assumed $K(t)$ to be some unknown linear operator. We remark that a structure can also be prescribed to $K(t)$, parameterized by a stencil $\psi$. Then, the kernel is $K(\psi(t))$, and the design variables now are $\theta = (W, \psi, b)$. Consequently, $K(\psi(t))$ can be thought of as a differential operator on the feature space, e.g. discrete Laplacian with a five point stencil. It then remains to compute the sensitivity of the Lagrangian functional w.r.t. $\psi$ to get the design equation. Note that this approach can further reduce the number of unknowns.*

Notice that so far the entire discussion has been at the continuous level and it has been independent of the number of network layers. Thus, it is expected that if we discretize (in time) the above optimality system, then the resulting gradient based solver is independent of the number of layers. We shall discretize the above optimality system in the next section.

## 4. Discrete fractional deep neural network

We shall adopt the *optimize-then-discretize* approach. Recall that the first order stationarity conditions for the continuous problem equation (12) are given by equation (14), equation (15), and equation (16). In order to discretize this system of equations, we shall first discuss the approximation of Caputo fractional derivative.

### 4.1. Approximation of Caputo derivative

There exist various approaches to discretize the fractional Caputo derivative. We will use the $L^1$-scheme (see [49, 62] and the references therein) to discretize the left and right Caputo fractional derivative $d_t^\gamma u(t)$ and $d_{T-t}^\gamma u(t)$ given in equation (3) and equation (4), respectively.

Consider the following fractional differential equation involving the **left** Caputo fractional derivative, for $0 < \gamma < 1$,

$$d_t^\gamma u(t) = f(u(t)), \quad u(0) = u_0. \tag{17}$$

We begin by discretizing the time interval $[0, T]$ uniformly with step size $\tau$,

$$0 = t_0 < t_1 < t_2 < \cdots < t_{j+1} < \cdots < t_N = T, \ \text{ where } \ t_j = j\tau.$$

Then, using the $L^1$-scheme, the discretization of equation (17) is given by: for $j = 0, \ldots, N-1$,

$$u(t_{j+1}) = u(t_j) - \sum_{k=0}^{j-1} a_{j-k}\left(u(t_{k+1}) - u(t_k)\right) + \tau^\gamma \Gamma(2-\gamma) f(u(t_j)), \tag{18}$$

where coefficients $a_{j-k}$ are given by,

$$a_{j-k} = (j+1-k)^{1-\gamma} - (j-k)^{1-\gamma}. \tag{19}$$

Next, let us consider the discretization of the fractional differential equation involving the **right** Caputo fractional operator, for $0 < \gamma < 1$,

$$d_{T-t}^\gamma u(t) = f(u(t)), \quad u(T) = u_T. \tag{20}$$

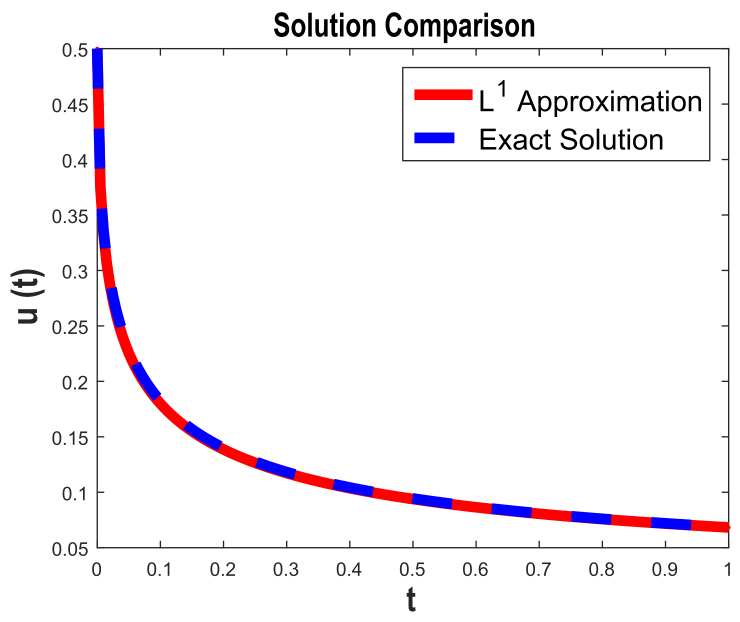

**Figure 1.** Comparison of the exact solution of equation (22) (*blue*) with an $L^1$ scheme approximation (*red*).

Again using $L^1$-scheme we get the following discretization of equation (20): for $j = N, ..., 1$,

$$u(t_{j-1}) = u(t_j) + \sum_{k=j}^{N-1} a_{k-j}\,(u(t_{k+1}) - u(t_k)) - \tau^\gamma \Gamma(2-\gamma) f(u(t_j)), \tag{21}$$

where coefficients $a_{k-j}$ are given, as before, by $a_{k-j} = (k+1-j)^{1-\gamma} - (k-j)^{1-\gamma}$. The example below illustrates a numerical implementation of the $L^1$-scheme equation (18).

**Example 4.1.** *Consider the linear differential equation*

$$d_t^{0.5} u(t) = -4\,u(t), \quad u(0) = 0.5. \tag{22}$$

*Then, the solution to equation* (22) *is given by, (see [59, section 42], also [55, section 1.2])*

$$u(t) = 0.5\,E_{0.5}(-4t^{0.5}),$$

*where* $E_\alpha$, *with* $0 < \alpha \in \mathbb{R}$, *is the Mittag Leffler function, see [55, Pg. 17], defined by*

$$E_\alpha(z) = E_{\alpha,1}(z) = \sum_0^\infty \frac{z^k}{\Gamma(\alpha k + 1)}.$$

*Figure 1 depicts the true solution and the numerical solution using discretization equation (18) for the above example with uniform step size* $\tau = 0.005$ *and final time,* $T = 1$.

## 4.2. Discrete optimality conditions

Next, we shall discretize the optimality conditions given in equations (14)–(16). Notice that each time-step corresponds to one layer of the neural network. It is necessary to do one forward propagation (state solve) and one backward propagation (adjoint solve) to derive an expression of the gradient with respect to the design variables.

(a) **Discrete state equation.** We use the $L^1$ scheme discussed in equation (18) to discretize the state equation (14) and arrive at

$$\begin{aligned} \overline{Y}(t_j) &= Y(t_{j-1}) - \sum_{k=1}^{j-1} a_{j-k}\,(Y(t_k) - Y(t_{k-1})) \\ &\quad + \tau^\gamma \Gamma(2-\gamma)\sigma(\overline{K}(t_{j-1})\overline{Y}(t_{j-1}) + \overline{b}(t_{j-1})), \quad j = 1, \ldots, N \\ \overline{Y}(t_0) &= Y_0. \end{aligned} \tag{24}$$

(b) **Discrete adjoint equation.** We use the $L^1$ scheme discussed in equation (21) to discretize the adjoint equation (15) and arrive at

$$\begin{aligned}\overline{P}(t_j) &= P(t_{j+1}) + \sum_{k=j+1}^{N-1} a_{k-j-1}\,\left(P(t_{k+1}) - P(t_k)\right) \qquad j = N-1, \dots, 0\\ &\quad - \tau^{\gamma}\Gamma(2-\gamma)\left[-\overline{K}(t_j)^{\intercal}\left(\overline{P}(t_{j+1}) \odot \sigma'\left(\overline{K}(t_j)\overline{Y}(t_{j+1}) + \overline{b}(t_j)\right)\right)\right],\\ \overline{P}(t_N) &= -\frac{1}{n}\overline{W}^{\intercal}\left(-C_{obs} + S(\overline{W}, \overline{Y}(t_N))\right).\end{aligned} \tag{25}$$

(c) **Discrete gradient w.r.t. design variables.** For $j = 0, \dots, N-1$, the approximation of the gradient in equation (16) with respect to the design variables is given by,

$$\begin{aligned}\nabla_W \mathcal{L}(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P}) &= \frac{1}{n}\left(-C_{obs} + S(\overline{W}, \overline{Y}(t_N))\right)\left(\overline{Y}(t_N)\right)^{\intercal}\\ &\quad + \nabla_W \mathcal{R}(\overline{W}, \overline{K}(t_N), \overline{b}(t_N)),\\ \nabla_K \mathcal{L}(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P}) &= -\overline{Y}(t_j)\left(\overline{P}(t_{j+1}) \odot \sigma'(\overline{K}(t_j)\overline{Y}(t_j) + \overline{b}(t_j))\right)^{\intercal}\\ &\quad + \nabla_K \mathcal{R}(\overline{W}, \overline{K}(t_j), \overline{b}(t_j)),\\ \nabla_b \mathcal{L}(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P}) &= -\langle \sigma'(\overline{K}(t_j)\overline{Y}(t_j) + \overline{b}(t_j)), \overline{P}(t_{j+1})\rangle_F\\ &\quad + \nabla_b \mathcal{R}(\overline{W}, \overline{K}(t_j), \overline{b}(t_j)).\end{aligned} \tag{26}$$

Whence, we shall create a gradient based method to solve the optimality condition equation (24)–equation (26). We reiterate that each computation of the gradient in equation (26), requires one state and one adjoint solve.

## 5. Fractional-DNN algorithm

Fractional-DNN is a supervised learning architecture, i.e. it comprises of a training phase and a testing phase. During the training phase, labeled data is passed into the network and the unknown parameters are learnt. Those parameters then define the trained Fractional-DNN model for that type of data. Next, a testing dataset, which comprises of data previously unseen by the network, is passed to the trained net, and a prediction of classification is obtained. This stage is known as the testing phase. Here the true classification is not shown to the network when a prediction is being made, but can later be used to compare the network efficiency, as we have done in our numerics. The three important components of the algorithmic structure are forward propagation, backward propagation, and gradient update. The forward and backward propagation structures are given in algorithms 1 and 2 and illustrated in figure 2 and figure 3, respectively. The gradient update is accomplished in the training phase, discussed in section 5.1. Lastly, the testing phase of the algorithm is discussed in section 5.2.

**Algorithm 1** Forward Propagation in Factional-DNN ($L^1$-scheme)

```
Input: $(Y_0, C_{obs}), W, \{K_j, b_j\}_{j=0}^{N-1}, N, \tau, \gamma$
Output: $\{Y_j\}_{j=1}^{N}, P_N,$
1: Let $z_0 = 0$.
2: for $j = 1, \cdots, N$ do
3:     for $k = 1, \cdots, j-1$ do
4:         Compute $a_{j-k}$:    {Use equation (19)}
5:         Update $z_k$:         $z_k = z_{k-1} + a_{j-k}\,(Y_k - Y_{k-1})$
6:     end for
7:     Update $Y_j$:             $Y_j = Y_{j-1} - z_{j-1} + (\tau)^{\gamma}\,\Gamma(2-\gamma)\,\sigma(K_{j-1}Y_{j-1} + b_{j-1})$
8: end for
9: Compute $P_N$:                $P_N = -(n)^{-1}\,W^{\intercal}(-C_{obs} + S(W, Y_N))$
```

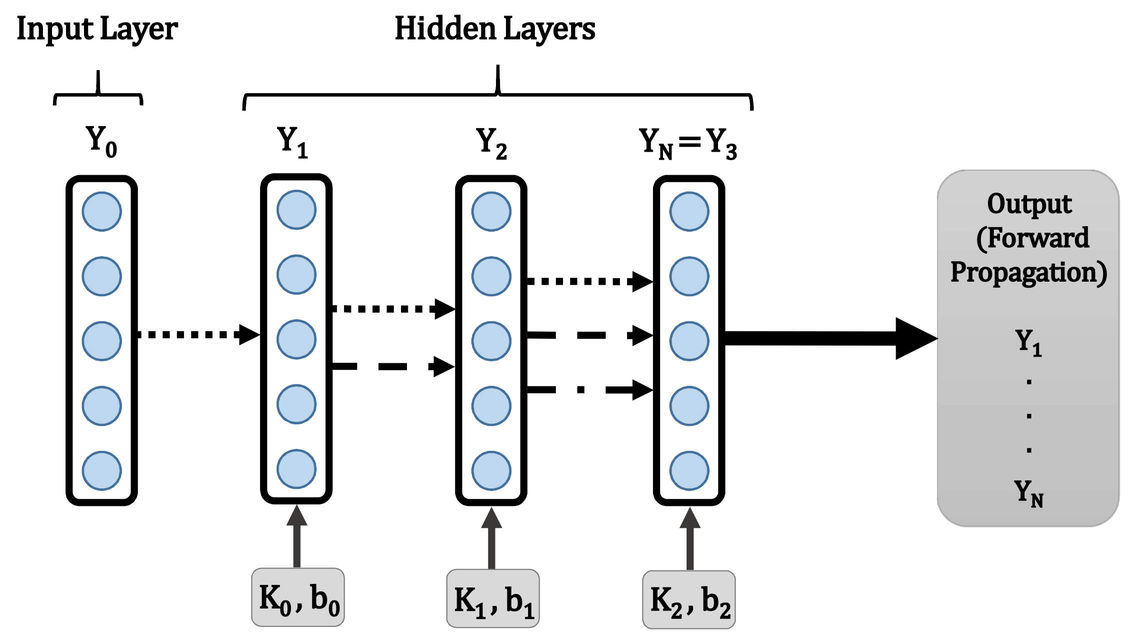

**Figure 2.** Representation of Forward Propagation in a 3-layer Fractional-DNN. Input data $Y_0$ is used to compute $Y_1$, along with respective design variables $(K_0, b_0)$. Next, both $Y_0$ and $Y_1$ (along with $(K_1, b_1)$) are used to compute $Y_2$. Finally, $Y_0$, $Y_1$ and $Y_2$ (along with $(K_2, b_2)$) are used to compute $Y_3$, which is the last layer of this network. Input transformation at each layer, $\{Y_i\}_{i=1}^{3}$, is stored, and passed to backward propagation. $Y_N$ is used for classification once the network has been trained. Forward propagation is equivalent to solving the state equation, forward in time. The style of the horizontal arrows (dotted, dashed, dot-dashed) represents how information from each respective layer is cascading forward. See algorithm 1 for details.

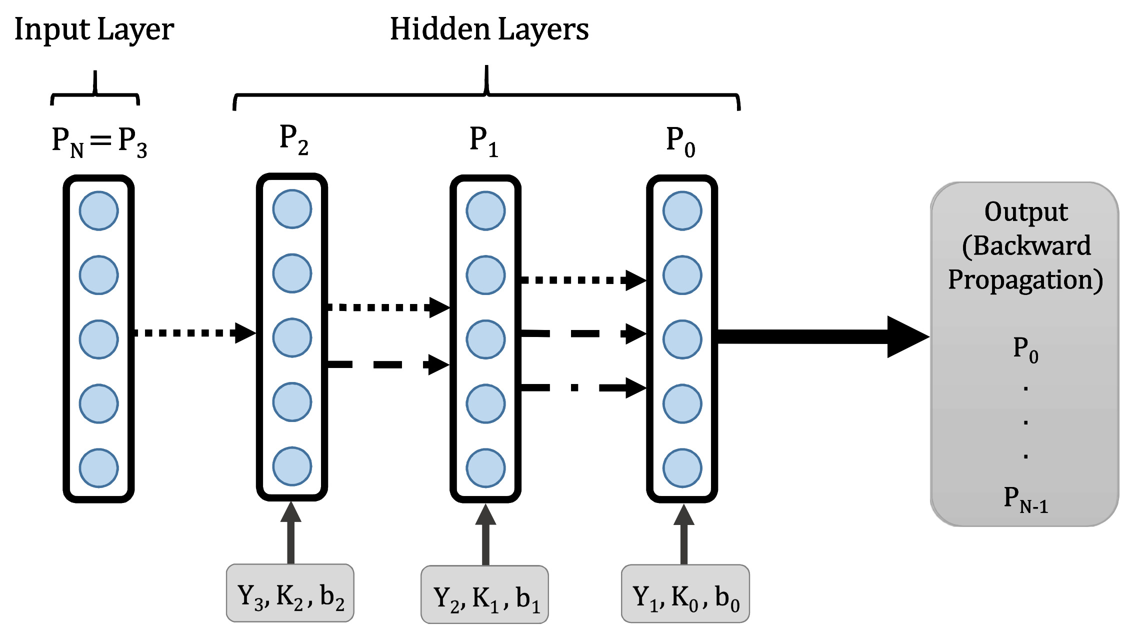

**Figure 3.** Representation of Backward Propagation in a 3-layer Fractional-DNN. Adjoint variable $P_N = P_3$, obtained using $Y_N$, is the input to the backward propagation. The network moves backward in time to compute $\{P_2, P_1, P_0\}$ using information from all the preceding layers. Adjoint variable transformation at each layer, $\{P_i\}_{i=1}^{3}$, is stored, and used later to compute the gradients w.r.t. the design variables. Backward propagation is equivalent to solving the adjoint equation, backward in time. The style of the horizontal arrows (dotted, dashed, dot-dashed) represents how information from each respective layer is cascading backwards. See algorithm 2 for details.

**Algorithm 2** Backward Propagation in Factional-DNN ($L^1$-scheme)

```
**Input:** $\{Y_j\}_{j=1}^N$, $P_N$, $\{K_j, b_j\}_{j=0}^{N-1}$, $N$, $\tau$, $\gamma$
**Output:** $\{P_j\}_{j=0}^{N-1}$
1: Let $x_0 = 0$.
2: for $j = N-1, \cdots, 0$ do
3:     for $k = j+1, \cdots, N-1$ do
4:         Compute $a_{k-j-1}$: {Use equation (19)}
5:         Compute $x_k$:    $x_k = x_{k-1} + a_{k-j-1}\,(P_{k+1} - P_k)$
6:     end for
7:     Update $P_j$:    $P_j = P_{j+1} + x_{N-1} - (\tau)^\gamma\,\Gamma(2-\gamma)\,[-K_j^\intercal(P_{j+1} \odot \sigma'(K_j Y_{j+1} + b_j))]$
8: end for
```

### 5.1. Training phase

The training phase of Fractional-DNN is shown in algorithm 3.

**Algorithm 3** Training Phase of Factional-DNN

```
**Input:** $(Y_0, C_{obs})$, $N$, $\tau$, $\gamma$, $m_1, m_2$
**Output:** $W$, $\{K_j, b_j\}_{j=0}^{N-1}$, $C_{train}, \alpha_{train}$,
1: Initialize $W, \{K_j, b_j\}_{j=0}^{N-1}$
2: for $i = 1, \cdots, m_1$
3:     Let $(\hat{Y}_0, \hat{C}_{obs}) \subset (Y_0, C_{obs})$ {Randomly select a mini-batch and apply BN using equation (27)}
4:     FORWARD PROPAGATION {Use algorithm 1 to get $\{\hat{Y}_j\}_{j=1}^N$, $P_N$}.
5:     BACKWARD PROPAGATION {Use algorithm 2 to get $\{P_j\}_{j=0}^{N-1}$}.
6:     GRADIENT COMPUTATION
7:     Compute $\nabla_W\mathcal{L}$, $\{\nabla_K\mathcal{L}\}$, $\{\nabla_b\mathcal{L}\}$
           $\nabla_W\mathcal{L} = (n)^{-1}\left(-C_{obs} + S(W, \hat{Y}_N)\right)(\hat{Y}_N)^\intercal + \nabla_W\mathcal{R}(W, K_j, b_j)$
           $\nabla_K\mathcal{L} = -\hat{Y}_j\left(P_{j+1} \odot \sigma'(K_j\hat{Y}_j + b_j)\right)^\intercal + \nabla_K\mathcal{R}(W, K_j, b_j)$
           $\nabla_b\mathcal{L} = -tr\left(\sigma'(K_j\hat{Y}_j + b_j)\,P_{j+1}\right) + \nabla_b\mathcal{R}(W, K_j, b_j)$
8:     Pass $\nabla_W\mathcal{L}$, $\nabla_K\mathcal{L}$, $\nabla_b\mathcal{L}$ to gradient based solver with $m_2$ max iterations to update $W, \{K_j, b_j\}_{j=0}^{N-1}$.
9:     Compute $\hat{C}_{train} = S(W, \hat{Y}_N)$
10:    Compare $\hat{C}_{train}$ to $\hat{C}_{obs}$ to compute $\alpha_{train}$
11: end for
```

### 5.2. Testing phase

The testing phase of Fractional-DNN is shown in algorithm 4.

**Algorithm 4 Testing Phase of Fractional-DNN**

```
**Input:** $\left(Y_0^{test}, C_{obs,test}\right)$, $W$, $\{K_j, b_j\}_{j=0}^{N-1}$, $N$, $\tau$, $\gamma$
**Output:** $C_{test}, \alpha_{test}$
  1. Let $Y_0 = Y_0^{test}$ {Apply BN using equation (27)}
  2. FORWARD PROPAGATION {Use algorithm 1 to get $\{Y_j\}_{j=1}^N$}.
  3. Compute $C_{test} = S(W, Y_N)$
  4. Compare $C_{test}$ to $C_{obs,test}$ to compute $\alpha_{test}$
```

**Remark 5.1** (Backward Propagation and Gradient Computation). *We remark that, unlike traditional machine learning architectures, the backward propagation in Fractional-DNN (algorithm 2) does not specifically compute the gradients of the loss function w.r.t. the unknowns $(K, b)$. It instead computes the adjoint variable P, in the spirit of an optimization algorithm. Information from forward and backward propagation is then used to compute the gradients $\nabla_K\mathcal{L}$ and $\nabla_b\mathcal{L}$. From practical implementation point of view, $\nabla_K\mathcal{L}$ and $\nabla_b\mathcal{L}$ can be computed inside the backward propagation routine for efficiency, where all the required information is available.*

### 5.3. Computational cost and storage

Recall that each iteration of the optimization solver (*line 2*, algorithm 3) calls for a forward propagation (algorithm 1) and backward propagation (algorithm 2), i.e. state and adjoint equation solve, to update the optimization variables. The fractional-DNN algorithm incurs the cost of solving a Fractional ODE instead of a standard time ODE, which is the cost incurred by the standard RNN [47].

Furthermore, similar to a classical RNN, fractional-DNN algorithm requires storage of $Y$ and $P$ from all layers in the network. The only difference between the two is the amount of information that needs to be accessed at each layer. During forward (or backward) propagation, a classical RNN only utilizes information from one previous layer, whereas the fractional-DNN uses information from all precedent layers in a cascading manner. This memory-laden architecture is in fact the unique feature of the fractional-DNN, which allows the algorithm to make more informed decisions as information propagates through layers in the network.

## 6. Numerical experiments

In this section, we present several numerical experiments where we use our proposed Fractional-DNN algorithm from section 5 to solve classification problems for two different datasets. We recall that the goal of classification problems, as the name suggests, is to classify objects into pre-defined class labels. First we prepare a training dataset and, along-with its classification, pass it to the training phase of Fractional-DNN (algorithm 3). This phase yields the optimal set of unknown parameters learned from the *training dataset*. They are then used to classify new data points from the *testing dataset* during the testing phase of Factional-DNN (algorithm 4). We compare the results of our Fractional-DNN with the classical RNN, see equation (9).

The rest of this section is organized as follows: First, we discuss some data preprocessing and implementation details. Then we describe the datasets being used, and finally we present the experimental results.

### 6.1. Implementation details

(a) **Batch normalization.** During the training phase, we use the batch normalization (BN) technique [63]. At each iteration we randomly select a mini-batch, $\hat{Y}_0 \subset Y_0$, which comprises of 50% of the training data. We then normalize the mini-batch to have a zero mean and a standard deviation of one, i.e.

$$\hat{Y}_0 = \frac{\hat{Y}_0 - \mu(\hat{Y}_0)}{s(\hat{Y}_0)}, \tag{27}$$

where $\mu$ is the mean and $s$ is the standard deviation of the mini-batch. The normalized mini-batch is then used to train the network for that iteration. At the next iteration, a new mini-batch is randomly selected. This process is repeated $m_2$ times. Batch normalization prevents gradient blow-up, helps speed up the learning and reduces the variation in parameters being learned.

Since the design variables are learnt on training data processed with BN, we also process the testing data with BN, in which case the mini-batch is the whole testing data.

(b) **Activation function.** For our experiments we use the hyperbolic tangent function as the activation function, for which,

$$\sigma(x) = \tanh(x), \text{ and } \sigma'(x) = 1 - \tanh^2(x).$$

(c) **Regularization.** For our experiments we use the following regularization,

$$\mathcal{R}(W,K,b) := \frac{\xi_W}{2}\|W\|_F^2 + \frac{\xi_K}{2\,N}\|(-\Delta)_h K(t)\|_F^2 + \frac{\xi_b}{2\,N}\|b(t)\|_2^2,$$

where $(-\Delta)_h$ is the discrete Laplacian, and $\xi_W, \xi_K, \xi_b$ are the scalar regularization strengths, and $\|\cdot\|_F$ is the Frobenius norm. It is well-known that a regularization helps to make the inverse problems well-posed. Notice that we have not carried out a rigorous analysis here, but in our numerical experiments, we observe that the performance of projected BFGS improves with regularization. The discrete Laplacian is approximated using the five-point stencil finite-difference method on a uniform grid, see, for instance, [64, section 3.2].

Notice that with the above regularization, we are enforcing Laplacian-smoothing on $K$. For a more controlled smoothness, one could also use the fractional Laplacian regularization introduced in [44], see also [2, 45].

(d) **Order of fractional time derivative.** For our computations, we choose $\gamma$ heuristically. We remark that this fractional exponent on time derivative can be learnt in a similar manner as the fractional exponent on Laplacian is learnt in [2], or as the authors do in [31].

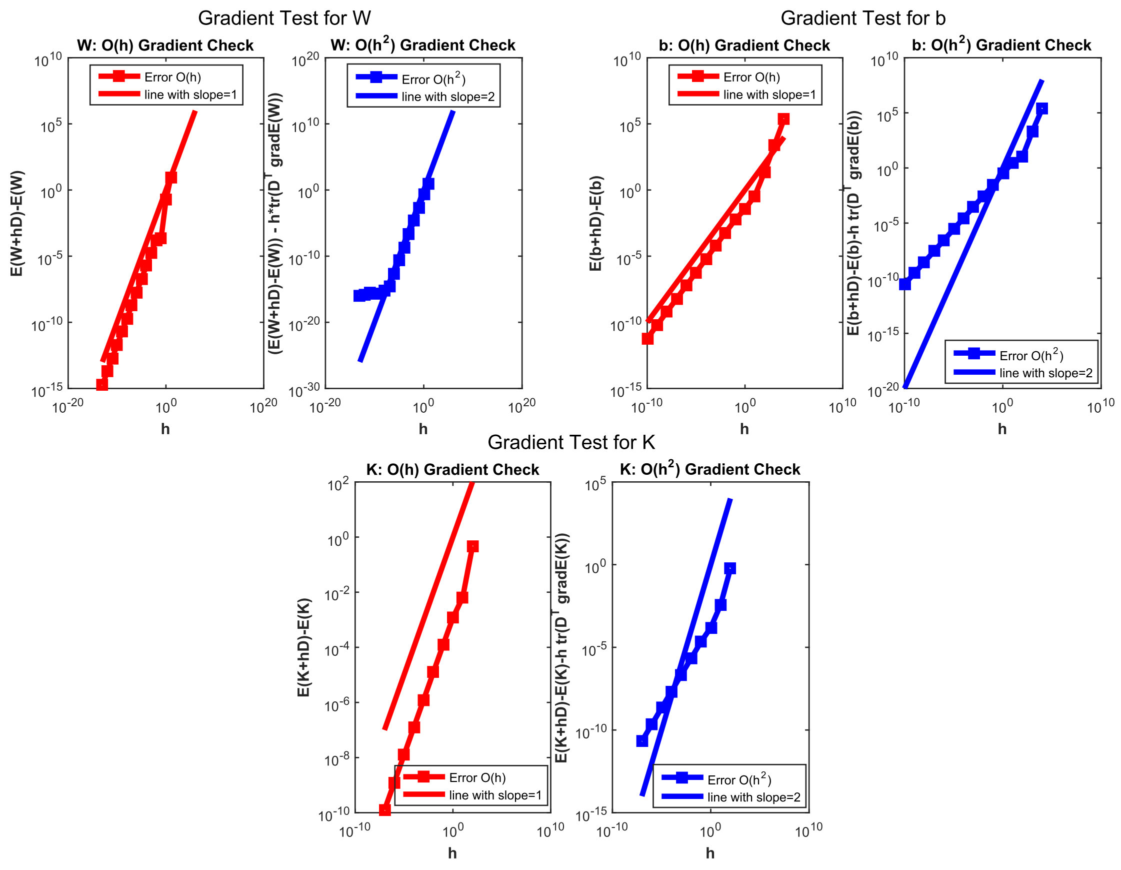

**Figure 4.** Comparison of discretized gradients with finite difference approximations. Note that the expected rate of convergence is obtained.

(e) **Optimization solver and Xavier initialization.** The optimization algorithm we use is the BFGS method with Armijo line search [65]. The stopping tolerance for the BFGS algorithm is set to $1\times 10^{-6}$ or maximum number of optimization iterations $m_2$, whichever is achieved first. However, in our experiments, the latter is achieved first in most cases. The design variables are initialized using Xavier initialization [11], according to which, the biases $b$ are initialized as 0, and the entries of $W$ and $K_j$ are drawn from the uniform distribution $U[-a,a]$. We consider $a=\sqrt{\frac{3}{n_f}}$ for the activation function $\sigma(\cdot)=\tanh(\cdot)$, and $a=\frac{1}{\sqrt{n_f}}$ for other activation functions.

(f) **Network layers vs. the final time.** For our experiments, we heuristically choose the number of layers $N$, and the discretization step-length for forward and backward propagation as $\tau=0.2$. Thus our final time is given by, $T=t_N=N\tau$.

(g) **Classification accuracy.** We remark that when we calculate $C_{train}=S(W,Y_N)$, we obtain a probability distribution of the samples belonging to the classes. We consider the class with the highest probability as the predicted class. Then, we use a very standard procedure to compare $C_{train}$ with $C_{obs}$.

$$\mathrm{n_{cor,train}} := \text{No. of correctly identified labels} = n - \frac{1}{2}\|C_{obs}-C_{train}\|_F^2,$$

$$\text{training error} = 1-\frac{\mathrm{n_{cor,train}}}{n}, \quad \text{and} \quad \alpha_{train}=\frac{\mathrm{n_{cor,train}}}{n}\times 100.$$

The same procedure is used to compute $C_{test}$ and $\alpha_{test}$.

(h) **Gradient test.** To verify the gradients in equation (26), we perform a gradient test by comparing them to a finite difference gradient approximation of system in equation (12). In figure 4 we show that the two conform and we obtain the expected order of convergence for all the design variables.

(i) **Computational platform.** All the computations have been carried out in MATLAB R2015b on a laptop with an Intel Core i7-8550U processor.

### 6.2. Experimental datasets

We describe the datasets we have used to validate our proposed Fractional-DNN algorithm below.

(a) **Dataset 1: Coordinate to Level Set (CLS).** This data comprises of a set of 2D coordinates, i.e. $Y_0 := \{(x_i, y_i) \mid i = 1, \cdots, n;\ (x_i, y_i) \in \mathbb{R}^2([0,1])\}$. Next, we consider the following piecewise function,

$$v(x,y) = \begin{cases} 1 & \forall\, x \leq y \\ 0 & \forall\, x > y \end{cases} \quad \forall\ x, y \in [0,1]. \tag{28}$$

The coordinates are the features in this case, hence $n_f = 2$. Further, we have $n_c = 2$ classes, which are the two level sets of $v(x,y)$. Thus, for the $i$th sample $Y_0^{(i)}$, $C_{obs}^{(i)} \in \mathbb{R}^{n_c}$ is a standard basis vector which represents the probability that $Y_0^{(i)}$ belongs to the class label $\{1, 2\}$.

(b) **Dataset 2: Perfume Data (PD) [66, 67].** This dataset comprises of odors of 20 different perfumes measured via a handheld meter (OMX-GR sensor) every second, for 28 seconds. For this data, $Y_0 := \{(x_i, y_i) \mid i = 1, \cdots, n;\ x_i, y_i \in \mathbb{Z}_+\}$, thus $n_f = 2$. The classes, $n_c = 20$, pertain to 20 different perfumes. we construct $C_{obs}$ in the same manner as we did for Dataset 1.

### 6.3. Forward propagation as a dynamical system

In the introduction we mentioned the idea of representing a DNN as an optimization problem constrained by a dynamical system. This has turned out to be a strong tool in studying the underlying mathematics of DNNs. In figure 5, we numerically demonstrate how this viewpoint enables a more efficient strategy for learning to distinguish between the classes. First we consider the perfume data, which has two features, namely the $(x, y)$ coordinates, and let it flow, i.e. forward propagate. When this evolved data is presented to the classifier function (e.g. softmax function in our case), a spatially well-separated data is easier to classify.

We illustrate this evolution in figure 5. We plot the input data $Y_0$ as well as the evolved data $Y_N$. The 20 different colors correspond to the 20 different classes of the Perfume Data, which help us visually track the evolution from $Y_0$ to $Y_N$. The evolution under standard RNN is shown in the *bottom left* plot, and that under Fractional-DNN is shown in the *bottom right* plot. The configuration for these plots is the same as discussed in section 6.5 below and pertains to the trained models. Notice that at the bottom right corner of the standard RNN evolution plot, the purple, pink and red data points are overlapping which pose a challenge for the classifier to distinguish among the classes. In contrast, Fractional-DNN has separated out those points quite well.

We remark that this separation also gives some intuition regarding the number of layers needed in a network. We need enough number of layers which would let the data evolve enough to be easily separable. However, this intuition via visualization can get restricted to $n_f \leq 3$. Therefore for data with $n_f > 3$, it may be challenging to get a sense of minimum number of layers needed to make the data separable-enough.

### 6.4. Vanishing gradient issue

In earlier sections, we remarked that Fractional-DNN handles the vanishing gradient issue in a better way. The vanishing gradient issue arises when the gradients w.r.t. the design variables vanishes across the layers as the network undergoes backward propagation, see [68] and references therein. As a consequence, feature extraction in the initial layers gets severely affected, which in turn affects the learning ability of the whole network. We illustrate this phenomenon for the networks under discussion in figure 6. In the *left* plot of figure 6, we compare the $\|\cdot\|_2$ of the gradient w.r.t. the design variables $\theta = (K, b)$ against optimization solver (steepest descent in this case) iterations for standard DNN (which does not have any skip connection) in *magenta*, classical RNN equation (9) in *black*, and Fractional-DNN with $L^1$-scheme approximation from algorithm 3 in red. In the *right* plot of figure 6 we have omitted the standard DNN plot to take a closer look at the other two. Observe that as gradient information propagates backward, i.e. from layer $N-1$ to 0, its magnitude reduces by one order in the case of standard RNN. This implies that enough information is not being passed to initial layers relative to the last layer. In contrast, the Fractional-DNN is carrying significant information back to the initial layers while maintaining a relative magnitude. This improves the overall health of the network and improves learning. This test is performed on Perfume Data (Dataset 2) with 70 layers and regularization turned off.

### 6.5. Experimental results

We now solve the classification problem equation (12) for the datasets described in section 6.2 via our proposed Fractional-DNN algorithm, presented in section 5. We then compare it with the standard RNN architecture given by system in equation (9). The details and results of our experiments are given in table 2.

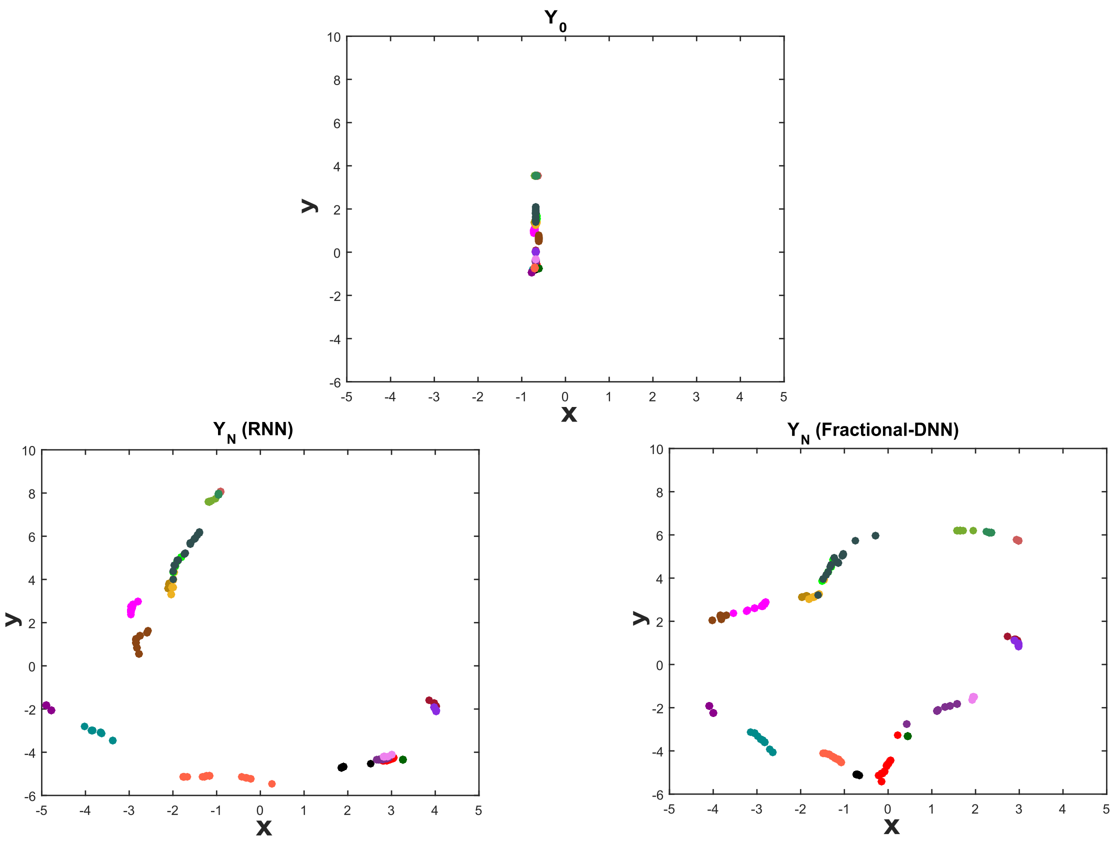

**Figure 5.** Forward propagation of perfume data from $Y_0$ (*Row 1*) to $Y_N$ via standard RNN (*Row 2, left*) and Fractional-DNN (*Row 2, right*). Note that the data is more linearly separable for Fractional-DNN. The various colors represent the respective classes.

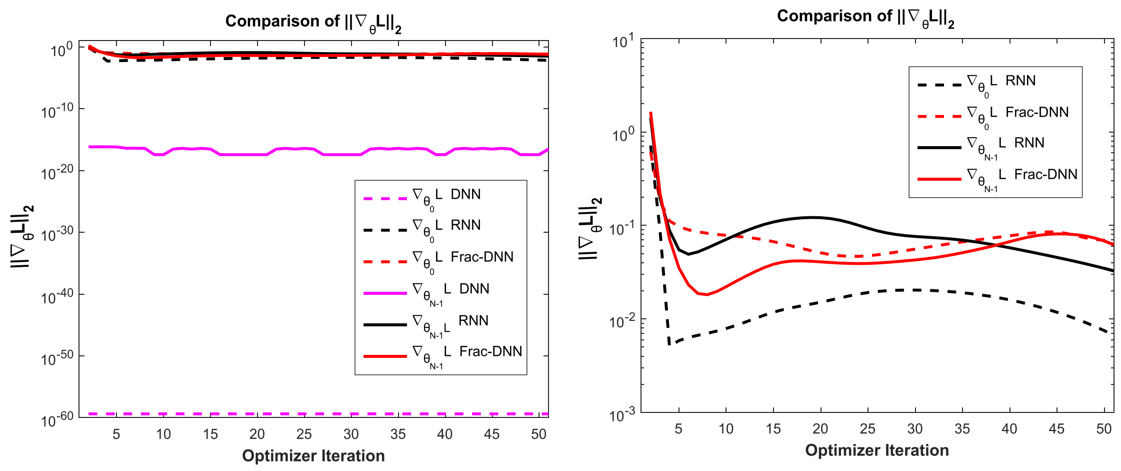

**Figure 6.** Demonstration of the gradient norm of $\theta = (K, b)$ at the first layer (dotted line) and last layer (solid line) of the network for various algorithms, namely standard DNN (magenta), RNN (black), and Fractional-DNN with $L^1$ scheme approximation (red). The figure on the *right* is the zoomed in version of figure on the *left*. Note the improvement in relative gradient propagation across layers for Fractional-DNN which leads to better learning ability and improves the vanishing gradient issue.

Note that the results obtained via Fractional-DNN are either comparable to (e.g. for CLS data) or significantly better than (e.g. for PD) the standard RNN architecture.

We remark that while CLS data (Dataset 1) is a relatively simpler problem to solve (two features and two classes), the Perfume Data (Dataset 2) is not. In the latter case, each dataset comprises of only two features, and there are 20 different classes. Furthermore, the number of available samples for training is small. In this sense, classification of this dataset is a challenging problem.

There have been some results on classification of perfume data using only the training dataset (divided between training and testing) [67], but to the best of our knowledge, classification on the complete dataset using both the training and testing sets [66] is not available.

**Table 2.** Comparison of classification accuracy for various datasets using the standard RNN equation (9) with our proposed Fractional-DNN equation (11) with $L^1$ scheme approximation. Note the improvement in results due to Fractional-DNN.

| Dataset<br>Time Derivative | CLS<br>Standard | CLS<br>Frac-$L^1$ | PD<br>Standard | PD<br>Frac-$L^1$ |
|---|---|---|---|---|
| $\gamma$ | – | 0.1 | – | 0.9 |
| $n_{train}$ | 10000 | 10000 | 560 | 560 |
| $n_{test}$ | 10000 | 10000 | 532 | 532 |
| $N$ | 5 | 5 | 35 | 35 |
| $m_1$ | 6 | 6 | 567 | 567 |
| $m_2$ | 30 | 30 | 15 | 15 |
| $\xi_W$ | $1 \times 10^{-1}$ | $1 \times 10^{-1}$ | $1 \times 10^{-8}$ | $1 \times 10^{-8}$ |
| $\xi_K$ | $1 \times 10^{2}$ | $1 \times 10^{2}$ | 0 | 0 |
| $\xi_b$ | $1 \times 10^{-2}$ | $1 \times 10^{-2}$ | 0 | 0 |
| $\alpha_{train}$ | 99.76% | 99.82% | 52.86% | 70.36% |
| $\alpha_{test}$ | 99.79% | 99.79% | 45.49% | 84.21% |

In our experiments, we have also observed that Fractional-DNN algorithm needs lesser number of Armijo line-search iterations than the standard RNN. This directly reflects an improvement in the learning rate via Fractional-DNN. We remark that in theory, Fractional-DNN should use memory more efficiently than other networks, as it encourages feature reuse in the network.

## 7. Discussion

There is a growing body of research which indicates that deep learning algorithms, e.g. a residual neural network, can be cast as optimization problems constrained by ODEs or PDEs. In addition, thinking of continuous optimization problems can make the approaches machine/architecture independent. This opens a plethora of tools from constrained optimization theory which can be used to study, analyze, and enhance the deep learning algorithms. Currently, the mathematical foundations of many machine learning models are largely lacking. Their success is mostly attributed to empirical evidence. Hence, due to the lack of mathematical foundation, it becomes challenging to fix issues, like network instability, vanishing and exploding gradients, long training times, inability to approximate non-smooth functions, etc. when a network does not perform as expected.

In this work we have developed a novel continuous model and stable discretization of deep neural network that incorporate history. In particular, we have developed a fractional deep neural network (Fractional-DNN) which allows the network to admit memory across all the subsequent layers. We have established this via an optimal control problem formulation of a deep neural network bestowed with a fractional time Caputo derivative. We have then derived the optimality conditions using the Lagrangian formulation. We have also discussed discretization of the fractional time Caputo derivative using $L^1$-scheme and presented the algorithmic framework for the discretization.

We expect that by keeping track of history in this manner improves the vanishing gradient problem and can potentially strengthen feature propagation, encourage feature reuse and reduce the number of unknown parameters. We have numerically illustrated the improvement in the vanishing gradient issue via our proposed Fractional-DNN. We have shown that Fractional-DNN is better capable of passing information across the network layers which maintains the relative gradient magnitude across the layers, compared to the standard DNN and standard RNN. This allows for a more meaningful feature extraction to happen at each layer.

We have shown successful application of Fractional-DNN for classification problems using various datasets, namely the Coordinate to Level Set (CLS dataset) and Perfume Data. We have compared the results against the standard-RNN and have shown that the Fractional-DNN algorithm yields improved results.

Fractional-DNN has a rigorous mathematical foundation and algorithmic framework which establishes a deeper understanding of deep neural networks with memory. This enhances their applicability to scientific and engineering applications.

We remark that code optimization is part of our forthcoming work. This would involve efficient Graphic Processing Unit usage and parallel computing capabilities. We also intend to develop a python version of this code and incorporate it into popular deep learning libraries like TensorFlow, PyTorch etc. We are also interested in expanding the efficiency of this algorithm to large-scale problems suitable for High Performance Computing.

## Acknowledgments

This work is partially supported by Department of Navy, Naval Postgraduate School - N00244-20-1-0005, Air Force Office of Scientific Research (AFOSR) under Award NO: FA9550-19-1-0036, and National Science Foundation grants DMS-1818772 and DMS-1913004. The second author is also partially supported by a Provost award at George Mason University under the Industrial Immersion Program.

The authors would like to thank Prasanna Balaprakash, Tamara G. Kolda and Lars Ruthotto for several discussions and comments during the course of this project.

## ORCID iDs

Harbir Antil https://orcid.org/0000-0002-6641-1449
Ratna Khatri https://orcid.org/0000-0003-0931-4025

## References

[1] He K, Zhang X, Ren S and Sun J 2016 Deep residual learning for image recognition *Proc. of the Conf. on Computer Vision and Pattern Recognition* pp 770–8
[2] Antil H, Di Z and Khatri R 2020 Bilevel optimization, deep learning and fractional Laplacian regularization with applications in tomography *Inverse Probl.* **36** 064001
[3] Wu S, Zhong Z and Liu Y 2018 Deep residual learning for image steganalysis *Multimed. Tools. Appl.* **77** 10437–53
[4] Jin K H, McCann M T, Froustey E and Unser M 2017 Deep convolutional neural network for inverse problems in imaging *IEEE Trans. Image Process.* **26** 4509–22
[5] Lee D, Yoo J, Tak S and Ye J C 2018 Deep residual learning for accelerated mri using magnitude and phase networks *IEEE Trans. Biomed. Eng.* **65** 1985–95
[6] Chen H, Dou Q, Yu L, Qin J and Heng P-A 2018 Voxresnet: Deep voxelwise residual networks for brain segmentation from 3d mr images *Neuroimage* **170** 446–55
[7] Hammernik K, Klatzer T, Kobler E, Recht M P, Sodickson D K, Pock T and Knoll F 2018 Learning a variational network for reconstruction of accelerated mri data *Magn. Reson. Med.* **79** 3055–71
[8] Tai Y, Yang J and Liu X 2017 Image super-resolution via deep recursive residual network *2017 Conf. on Computer Vision and Pattern Recognition (CVPR)* pp 2790–2798
[9] Zhang Q, Yuan Q, Zeng C, Li X and Wei Y 2018 Missing data reconstruction in remote sensing image with a unified spatial–temporal–spectral deep convolutional neural network *IEEE Trans. Geosci. Remote Sens.* **56** 4274–88
[10] Bischke B, Bhardwaj P, Gautam A, Helber P, Borth D and Dengel A 2017 Detection of flooding events in social multimedia and satellite imagery using deep neural networks *Working Notes Proc. of the MediaEval 2017 MediaEval Benchmark*, September 13-15 Dublin, Ireland MediaEval
[11] Glorot X and Bengio Y 2010 Understanding the difficulty of training deep feedforward neural networks *Proc. of the Thirteenth Int. Conf. on Artificial Intelligence and Statistics* vol 9 *Proc. of Machine Learning Research* Chia Laguna Resort, Sardinia, Italy 13–15 May PMLR pp 249–256
[12] Qiu J, Wu Q, Ding G, Xu Y and Feng S 2016 A survey of machine learning for big data processing *EURASIP J. Adv. Signal Process.* **2016** 67
[13] Wigderson A 2019 *Mathematics and Computation. A Theory Revolutionizing Technology and Science* (Princeton, NJ: Princeton University Press)
[14] Ruthotto L and Haber E 2019 Deep neural networks motivated by partial differential equations *J. Math. Imaging Vis.* **62** 352–64
[15] Weinan E 2019 Machine learning: Mathematical theory and scientific applications *Not. Am. Math. Soc.* **66** 1813–20
[16] Goldt S, Mézard M, Krzakala F and Zdeborová L 2019 Modelling the influence of data structure on learning in neural networks arXiv:1909.11500
[17] Mallat S and Waldspurger I 2013 Deep learning by scattering arXiv:1306.5532
[18] Bengio Y, Simard P and Frasconi P 1994-03 Learning long-term dependencies with gradient descent is difficult *IEEE Trans. Neural Netw.* **5** 157–166
[19] Veit A, Wilber M J and Belongie S 2016 Residual networks behave like ensembles of relatively shallow networks *Adv. Neural Inf. Process. Syst. 29* Lee D D, Sugiyama M, Luxburg U V, Guyon I and Garnett R Curran Associates, Inc. 550–8
[20] Chang B, Meng L, Haber E, Tung F and Begert D 2017 Multi-level residual networks from dynamical systems view arXiv:1710.10348
[21] Huang G, Liu Z, van der Maaten L and Weinberger K Q 2017 Densely connected convolutional networks *Proc. of the Conf. on Computer Vision and Pattern Recognition* pp 2261–9
[22] Zhang Y, Tian Y, Kong Y, Zhong B and Fu Y 2018 Residual dense network for image super-resolution *2018 IEEE/ Conf. on Computer Vision and Pattern Recognition* (https://doi.org/10.1109/cvpr.2018.00262)
[23] Srivastava R K, Greff K and Schmidhuber J 2015 Training Very Deep Networks *Advances in Neural Information Processing Systems 28* (Curran Associates, Inc.) Cortes C, Lawrence N D, Lee D D, Sugiyama M and Garnett R
[24] Cortes C, Gonzalvo X, Kuznetsov V, Mohri M and Yang S 2016 Adanet: Adaptive structural learning of artificial neural networks *Efficient Methods for Deep Neural Networks (EMDNN)* pp 874–83
[25] Chen K, Chen K, Wang Q, He Z, Hu J and He J 2019 Short-term load forecasting with deep residual networks *IEEE Trans. Smart Grid* **10** 3943–52
[26] Imaizumi M and Fukumizu K 2019 Deep neural networks learn non-smooth functions effectively *Proc. of Machine Learning Research* arXiv:1802.04474 PMLR vol 89 pp 869–78
[27] Haber E and Ruthotto L 2017 Stable architectures for deep neural networks *Inverse Probl.* **34** 014004
[28] Lu Y, Zhong A, Li Q and Dong B 2017 Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations *Proc. of Machine Learning Research* **80** 3276–85
[29] Schöenlieb C B, Benning M, Ehrhardt M, Owren B and Celledoni E 2019 Dataset Research data supporting 'deep learning as optimal control problems' University of Cambridge

[30] Benning M, Celledoni E, Ehrhardt M, Owren B and Schönlieb C-B 2019 Deep learning as optimal control problems: Models and numerical methods *J. Comput. Dyn.* **6** 171–98
[31] Pang G, Lu L and Karniadakis G E 2019 fpinns: Fractional physics-informed neural networks *SIAM J. Sci. Comput.* **41** A2603–A2626
[32] Raissi M, Perdikaris P and Karniadakis G E 2019 Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations *J. Comput. Phys.* **378** 686–707
[33] Gulian M, Raissi M, Perdikaris P and Karniadakis G E 2019 Machine learning of space-fractional differential equations *SIAM J. Sci. Comput.* **41** A2485–A2509
[34] Zúñiga-Aguilar C J, Romero-Ugalde H M, Gómez-Aguilar J F, Escobar-Jiménez R F and Valtierra-Rodríguez M 2017 Solving fractional differential equations of variable-order involving operators with mittag-leffler kernel using artificial neural networks *Chaos Solitons Fractals* **103** 382–403
[35] Zú niga-Aguilar C J, Coronel-Escamilla A, Gómez-Aguilar J F, Alvarado-Martínez V M and Romero-Ugalde H M 2018 New numerical approximation for solving fractional delay differential equations of variable order using artificial neural networks *Eur. Phys. J. Plus* **133** 75
[36] Silling S A 2000 Reformulation of elasticity theory for discontinuities and long-range forces *J. Mech. Phys. Solids* **48** 175–209
[37] Seleson P, Parks M L, Gunzburger M and Lehoucq R B 2009 Peridynamics as an upscaling of molecular dynamics *Multiscale Model. Simul.* **8** 204–27
[38] Metzler R and Klafter J 2000 The random walk's guide to anomalous diffusion: a fractional dynamics approach *Phys. Rep.* **339** 1–77
[39] Antil H, Khatri R and Warma M 2019 External optimal control of nonlocal PDEs *Inverse Probl.* **35** 084003
[40] Antil H, Verma D and Warma M 2020 External optimal control of fractional parabolic PDEs *ESAIM Control Optim. Calc. Var.* **26** 4065621 arXiv:1904.07123
[41] Atangana A 2018 Non validity of index law in fractional calculus: A fractional differential operator with markovian and non-markovian properties *Physica* A **505** 688–706
[42] Weiss C J, van Bloemen Waanders B G and Antil H 2020 Fractional operators applied to geophysical electromagnetics *Geophys. J. Int.* **220** 1242–59
[43] Antil H, D'Elia M, Glusa C A, Weiss C J and van Bloemen Waanders B G 2019 A fast solver for the fractional helmholtz equation *Technical report*, Sandia National Lab. (SNL-NM) Albuquerque, NM (United States)
[44] Antil H and Bartels S 2017 Spectral approximation of fractional PDEs in image processing and phase field modeling *J. Comput. Methods Appl. Math.* **17** 661–78
[45] Antil H and Rautenberg C N 2019 Sobolev spaces with non-Muckenhoupt weights, fractional elliptic operators and applications *SIAM J. Math. Anal.* **51** 2479–503
[46] Brown T, Du S, Eruslu H and Sayas F-J 2018 Analysis of models for viscoelastic wave propagation *Appl. Math. Nonlinear Sci.* **3** 55–96 arXiv:1802.00825
[47] Günther S, Ruthotto L, Schroder J, Cyr E and Gauger N 2020 Layer-parallel training of deep residual neural networks *SIAM J. Math. Data Sci.* **2** 1–23
[48] Antil H, Lizama C, Ponce R and Warma M 2019 Convergence of solutions of discrete semi-linear space-time fractional evolution equations arXiv:1910.07358
[49] Antil H, Otárola E and Salgado A J 2016 A space-time fractional optimal control problem: analysis and discretization *SIAM J. Control Optim.* **54** 1295–328
[50] Thiao A and Sene N 2020 Fractional optimal economic control problem described by the generalized fractional order derivative *4th Int. Conf. on Computational Mathematics and Engineering Sciences (CMES-2019)* eds Dutta H, Hammouch Z, Bulut H and Baskonus H M (Cham: Springer Int. Publishing) pp 36–48
[51] Jajarmi A, Ghanbari B and Baleanu D 2019 A new and efficient numerical method for the fractional modeling and optimal control of diabetes and tuberculosis co-existence *Chaos* **29** 093111
[52] Bengio Y 2012 *Practical Recommendations for Gradient-Based Training of Deep Architectures* (Berlin: Springer) pp 437–478
[53] Zú niga-Aguilar C, Gómez-Aguilar J F, Alvarado-Martínez V M and Romero-Ugalde H M 2020 Fractional order neural networks for system identification *Chaos Solitons Fractals* **130** 109444
[54] Kilbas A A, Srivastava H M and Trujillo J J 2006 *Theory and Applications of Fractional Differential Equations* (Amsterdam: Elsevier) vol 204 North-Holland Mathematics Studies
[55] Podlubny I 1999 Fractional differential equations: an introduction to fractional derivatives, fractional differential equations, to methods of their solution and some of their applications *Mathematics in Science and Engineering* (London: Academic)
[56] Nocedal J and Wright S 2006 Numerical optimization *Springer Series in Operations Research and Financial Engineering* (New York: Springer Science and Business Media)
[57] Anderson D G 1965 Iterative procedures for nonlinear integral equations *J. ACM* **12** 547–60
[58] Roux N L, Schmidt M and Bach F R 2012 A stochastic gradient method with an exponential convergence rate for finite training sets *Adv. Neural Inform. Process. Syst.* pp 2663–2671
[59] Samko S G, Kilbas A A and Marichev O I 1993 *Fractional Integrals and Derivatives* (Yverdon: Gordon and Breach Science Publishers)
[60] Scherer R 2020 *Computer Vision Methods for Fast Image Classification and Retrieval* (Berlin: Springer)
[61] Cireşan D, Meier U, Masci J and Schmidhuber J 2012 Multi-column deep neural network for traffic sign classification *Neural Netw.* **32** 333–8 Selected Papers from IJCNN 2011
[62] Jin B, Lazarov R and Zhou Z 2019 Numerical methods for time-fractional evolution equations with nonsmooth data: a concise overview *Comput. Methods Appl. Mech. Eng.* **346** 332–58
[63] Ioffe S and Szegedy C 2015 Batch normalization: Accelerating deep network training by reducing internal covariate shift *Proc. of the 32nd Int. Conf. on Int. Conf. on Machine Learning - Volume 37* ICML'15 JMLR.org pp 448–456
[64] LeVeque R J 2007 *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems* (Philadelphia, PA: SIAM)
[65] Kelley C T 1999 Iterative methods for optimization *Front. Appl. Math.* (Philadelphia, PA: SIAM)
[66] Dua D and Graff C 2017 UCI machine learning repository
[67] Esme E and Karlik B 2016 Fuzzy c-means based support vector machines classifier for perfume recognition *Appl. Soft Comput.* **46** 452–8
[68] Goh G B, Hodas N O and Vishnu A 2017 Deep learning for computational chemistry *J. Comput. Chem.* **38** 1291–307