An optimal time variable learning framework for Deep Neural Networks[*]

Harbir Antil, Hugo Díaz, and Evelyn Herberg

Feature propagation in Deep Neural Networks (DNNs) can be associated to nonlinear discrete dynamical systems. The novelty, in this paper, lies in letting the discretization parameter (time step-size) vary from layer to layer, which needs to be learned, in an optimization framework. The proposed framework can be applied to any of the existing networks such as ResNet, DenseNet or Fractional-DNN. This framework is shown to help overcome the vanishing and exploding gradient issues. Stability of some of the existing continuous DNNs such as Fractional-DNN is also studied. The proposed approach is applied to an ill-posed 3D-Maxwell's equation.

AMS 2000 subject classifications: 34A08, 49J15, 68T05, 82C32.
Keywords and phrases: Deep learning, Deep Neural Network, fractional time derivatives, fractional neural network, residual neural network, optimal network architecture, exploding gradients, vanishing gradients.

1. Introduction

Consider a network architecture, for example, residual neural network (ResNet)

$$(1.1) \qquad y^{[\ell]} = y^{[\ell-1]} + \tau\sigma(y^{[\ell-1]}, \theta^{[\ell-1]}),$$

which contains a hyperparameter τ. It is also possible to consider other architectures such as feedforward networks etc. The above neural network can be understood as the time-discretization of a non-linear ordinary differential equation (ODE). The feature vector $y^{[\ell]}$ is computed by forward propagation from the previous layers feature vector $y^{[\ell-1]}$ and network parameters,

which are collected in $\theta^{[\ell-1]}$, using an activation function σ. In most of the existing literature, τ is a given fixed constant. The main novelty of this work lies in

$$\text{replacing } \tau \text{ by learning variables } \tau^{[\ell]}$$

and the treatment of these $\tau^{[\ell]}$. These variables can be understood as the 'time step-sizes' in Deep Neural Networks (DNNs). This paper considers them as optimization variables, not as hyperparameters. Furthermore, the parameters $\tau^{[\ell]}$ are allowed to differ from layer to layer, i.e., time grid can be non-equidistant. Notice that this τ-variable framework can be applied to any of the existing networks of type (1.1). The deep learning optimization problem will now also learn optimal parameters $\tau^{[\ell]}$ in addition to the standard DNN parameters. As will be illustrated throughout the article, the presented approach is not just a scaling of the activation function σ by $\tau^{[\ell]}$. This will become evident when applying the proposed framework to Fractional-DNNs, where $\tau^{[\ell]}$ enters in multiple ways, see Remark 5.1. For a PyTorch implementation of τ-variable DNNs, we refer to [7].

This proposed approach presents multiple advantages, including:

- The proposed framework leads to *optimal adaptive time discretizations of DNNs*, such as ResNets and Fractional-DNNs, tailored to the optimization/learning problem. This is different than time adaptive approaches in [16, 10] where adaptivity is tailored to the forward problem (1.1) but not the learning (optimization) problem.
- The proposed framework can further help (as is rigorously established) overcome the *vanishing and exploding gradient* problems in networks such as ResNets and Fractional-DNNs. Notice, that motivation behind introducing ResNets [34, 33] was vanishing gradients and Fractional-DNNs was vanishing and exploding gradients [5]. Fractional derivatives are nonlocal in nature, therefore solution at current time step require information about solution at all the previous time steps, i.e., the system has memory effects. Additionally, fractional derivatives have less smoothing requirements. For a complete theoretical analysis of fractional DNN, see [1].
- If $\tau^{[\ell]}$ for any layer ℓ is close to zero, then the associated layer is redundant and can be deleted without sacrificing the accuracy. Thus leading to small yet accurate DNNs.
- Variable $\tau^{[\ell]}$ helps improve the training error decay, see Figure 1.

The main idea to approximate parameterized PDEs and solve the inverse problems using Fractional-DNNs has been recently introduced in [3].
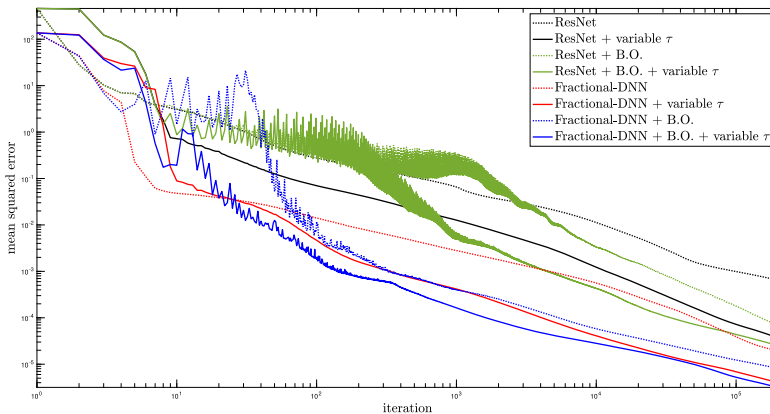
Figure 1: The panel shows the mean squared error during training when the variable-τ framework is applied to a ResNet and a Fractional DNN for an ill-posed 3D-Maxwell's equation. More details are available in Section 7.

The present article not only introduces the aforementioned variable time step framework, but for the first time, to the best of our knowledge, also applies the DNNs, such as ResNets and Fractional-DNNs, to ill-posed problems such as Maxwell's equations with Gauss's law. A comparison of these standard DNNs with their time-step variable versions has also been carried out. The problem is ill-posed in the following sense: Nédélec finite elements are traditionally used to discretize Maxwell's equations. However, they are curl-conforming and thus the Gauss's law (divergence condition) cannot be directly imposed [17]. The DNNs are shown to generalize well on the unseen data and physical domains.

Deep learning is a nascent field of research with many exciting applications, for example imaging science [5, 33, 39, 51, 62], biomedical applications [14, 30, 42], satellite imagery, remote sensing [11, 59, 64], segmentation [53], and gaming [56]. Recently, this topic is starting to receive significant attention from mathematicians [19, 23]. Especially, the fact that DNNs of type (1.1) can be viewed as optimization problems constrained by discrete dynamical systems [5, 10, 13, 28, 29, 47, 55] and partial differential equations [46, 54]. In these settings, the state-of-the-art is to consider an equidistant time grid, where the time step-size τ is chosen before running the optimization algorithm to identify weights, i.e. τ is a hyperparameter.

The articles [31, 38] suggest to consider $\tau^{[\ell]}$ as hyperparameters for physics-informed neural networks and ResNets, respectively. In both cases the hyperparameters $\tau^{[\ell]}$ are seen as scalings applied, outside [31] and inside

[38], the activation function. We also refer to [32] where parameterized ReLU was first introduced. In [38], a global $\tau = \tau^{[\ell]}$ for all ℓ is considered leading to training error improvement and more accurate solutions, which will also hold true for our more general setting. In [31], a sequence of $\tau^{[\ell]}$, fulfilling a probabilistic condition, is chosen to avoid exploding gradients. In contrast, we let the optimization algorithm learn $\tau^{[\ell]}$ and provide deterministic arguments to overcome the vanishing and exploding gradient problems.

Outline. This article is organized as follows. Section 2 introduces some basic preliminary results and notation. We introduce definitions of fractional derivatives and state a generic DNN. We also describe an extension of this DNN to include a recently introduced bias ordering idea from [2] which offers multiple advantages such as narrowing the parameter search space. This is followed by Section 3 where the relation between continuous DNNs and dynamical systems is considered. Special attention is given to two continuous version of DNNs: ResNet (DNN with standard time derivative) and Fractional-DNN (DNN with fractional time derivative). Stability results for these two DNNs are also provided. Notice, that Fractional-DNNs have been recently introduced in [5] for classification and further extended in [3] to inverse problems with PDEs. They have multiple advantages over ResNets as they can incorporate memory into the network due to the nonlocal nature of fractional derivatives and due to the low regularity requirements of fractional derivatives, they can be applied to non-smooth functions. As stated above, one of the main motivations behind introducing Fractional-DNN was to overcome vanishing and exploding gradients. The article [5] provides numerical evidence of overcoming the vanishing gradient problem.

Next, in Section 4 we state selected DNN architectures and compare them for a fixed τ. The new framework with variable $\tau^{[\ell]}$ is applied to the architectures of Section 4 in Section 5. Notice that our approach is broad and can be applied to any network architecture of type (1.1) and it is independent of the choice of the loss functional. Clearly, the proposed approach inherits all the positive aspects of these existing networks. Additionally, the new framework is rigorously shown to overcome vanishing and exploding gradient issues (cf. Section 6).

Finally, in Section 7 we illustrate the efficacy of our approach with the help of an ill-posed 3D-Maxwell's equation. The numerical examples validate the above mentioned advantages of the proposed framework. In particular, stability and network reduction.

2. Preliminaries

The goal of this section is to introduce the relevant notation and abstract optimization problems arising while training the DNNs. The content of this section is well-known [5, 3, 2].

Symbol	Description
$L \in \mathbb{N}$	Number of network layers (i.e. network depth)
$N \in \mathbb{N}$	Number of distinct data samples
n_ℓ	Number of nodes in layer ℓ
$y^{[\ell]} \in \mathbb{R}^{n_\ell}$	Feature vector in layer ℓ
σ	Activation function
$W^{[\ell]} \in \mathbb{R}^{n_{\ell+1} \times n_\ell}$	Weights in layer ℓ
$b^{[\ell]} \in \mathbb{R}^{n_{\ell+1}}$	Biases in layer ℓ
$\tau^{[\ell]} \in \mathbb{R}$	Time step-size in layer ℓ
$\theta^{[\ell]} \in \mathbb{R}^{n_{\ell+1}(n_\ell+1)+1}$	Vector of all variables (weights, biases and time step-size) in layer ℓ
P_j^ℓ	Projection matrix from layer j onto layer ℓ
ϕ	Adjoint variables
$\mathcal{F}$	Network represented as a function
f_ℓ	Layer function
J	Loss function
$\lambda_1, \lambda_2, \beta \in \mathbb{R}$	Regularization parameters
$\mathcal{L}$	Lagrangian
$\{u, S(u)\}$	Input / Output pair of training data
$\Gamma(\cdot)$	Euler's Gamma function

2.1. Caputo fractional derivative

In preparation for the Fractional-DNN architecture, we next introduce the left and right Caputo fractional derivatives for absolutely continuous functions and refer to [4, Definitions 2.1 and 2.4, and Proposition 2.3] and [41, (2.4.17) and (2.4.18)] for details.

Definition 2.1 (Left Caputo Fractional Derivative). Let $y \in W^{1,1}([0, T]; X)$, with X denoting a Banach space. The left Caputo fractional derivative of order $\gamma \in (0, 1)$ is given by

$$(2.1) \qquad \partial_t^\gamma y(t) = c_\gamma \int_0^t \frac{y'(r)}{(t-r)^\gamma} dr,$$

where $c_\gamma := \frac{1}{\Gamma(1-\gamma)}$ and $\Gamma(\cdot)$ is Euler's Gamma function.

Definition 2.2 (Right Caputo Fractional Derivative). Let $y \in W^{1,1}([0,T]; X)$, with X denoting a Banach space. The right Caputo fractional derivative of order $\gamma \in (0, 1)$ is given by

$$\partial_{T-t}^\gamma y(t) = -c_\gamma \int_t^T \frac{y'(r)}{(r-t)^\gamma} dr.$$

Next, we introduce the general deep learning problem as an optimization problem with DNN constraints.

2.2. Deep learning problem

Consider a neural network architecture with an input layer of dimension n_0, $L - 1$ hidden layers of dimension n_ℓ for $\ell = 1, \ldots, L - 1$ and an output layer of dimension n_L. Then we can represent this network as a function

$$(2.2) \qquad \mathcal{F} = f_{L-1} \circ f_{L-2} \circ \cdots \circ f_0,$$

where $\{f_\ell\}_{\ell=0}^{L-1}$ are the layer functions. These layer functions are parameterized by weight matrices $W^{[\ell]} \in \mathbb{R}^{n_{\ell+1} \times n_\ell}$ and bias vectors $b^{[\ell]} \in \mathbb{R}^{n_{\ell+1}}$. The definition of f_ℓ depends on the network architecture. We will introduce different options in Section 4.

The weights and biases are identified during a training process that requires solving an optimization problem. Let $\{u^{(i)}, S(u^{(i)})\}_{i=1}^N$ (input/ output pairs) denote the training data. Then the goal is to match the output of the DNN with the data points $S(u^{(i)})$. This is accomplished by minimizing a loss functional J and the resulting optimization problem is given by:

$$(2.3) \qquad \min_{\{W^{[\ell]}\}_{\ell=0}^{L-1}, \{b^{[\ell]}\}_{\ell=0}^{L-2}} J\left(\{(y^{[L](i)}, S(u^{(i)}))\}_i, \{W^{[\ell]}\}_\ell, \{b^{[\ell]}\}_\ell\right)$$

$$\text{subject to} \qquad y^{[L](i)} = \mathcal{F}\left(u^{(i)}; (\{W^{[\ell]}\}_\ell, \{b^{[\ell]}\}_\ell)\right) \quad i = 1, \ldots, N.$$

One standard choice for the loss function is the mean squared error

$$J := \frac{1}{2N} \sum_{i=1}^N \|y^{[L](i)} - S(u^{(i)})\|_2^2.$$

Another example is the cross-entropy, see for more examples [26]. The choice of J is dictated by the application. For our cause, it is not relevant which

of these options is chosen, since our focus is on DNNs represented by the operator $\mathcal{F}$.

It is also common to add regularization, for example, ℓ^1 and ℓ^2 regularizations on weights and biases

$$(2.4) \quad J_{\lambda_1} = J + \frac{\lambda_1}{2} \sum_{\ell=0}^{L-1} \left(\|W^{[\ell]}\|_2^2 + \|W^{[\ell]}\|_1 \right) + \frac{\lambda_1}{2} \sum_{\ell=0}^{L-2} \left(\|b^{[\ell]}\|_2^2 + \|b^{[\ell]}\|_1 \right),$$

where $\lambda_1 > 0$ is the regularization parameter. Notice that the 2-norms for the weight matrices are in fact Frobenius norms.

Before, we continue, we emphasize that recently, the article [2] has extended the above generic network (2.3) by incorporating ordering among the bias vector components in each layer. The main idea is that in each layer ℓ, with $\ell = 0, \ldots, L - 2$, one enforces

$$(2.5) \qquad\qquad b_j^{[\ell]} \le b_{j+1}^{[\ell]}, \quad j = 1, \ldots, n_{\ell+1} - 1,$$

where the subscript j indicates the bias vector component. This approach offers multiple advantages as highlighted in [2]. Our numerical examples further provides a comparison between with and without bias ordering framework. Notice that the bias ordering is implemented using a penalty framework, see [2] for details.

Next, we provide a mathematical background behind learning the time step-sizes $\tau^{[\ell]}$. To start, we discuss a link between some DNNs and dynamical systems.

3. Continuous DNNs

In this section, we study the continuous structure of multiple DNNs, cf. (1.1) and (2.2). This section mainly focuses on the stability of these architectures, which will follow from a connection with dynamical systems. For other approaches we refer to [8, 31] and references therein. Since we are primarily interested in stability results, we start with a basic remark on a (finite) network with a Lipschitz activation function, like ReLU. The finite composition of Lipschitz functions is also a Lipschitz function, and therefore differentiable almost everywhere by Rademacher's theorem. In the following we show some historical connections between neural networks and dynamical systems. Later on we consider continuous Fractional-DNNs make use of fractional derivatives. These derivatives are nonlocal operators and one notices that solution at a current time depends on the entire history, i.e., fractional derivatives enable memory into the DNN.

3.1. Ordinary differential equations and neural networks

The relation between Neural Networks (NNs) and differential equations is not new. In fact, in the late '80s, in [50] the following model was considered for the activity of j-th neuron:

$$(3.1) \qquad \frac{dy_j}{dt} = -\alpha y_j + \beta\sigma\Big(\sum_k w_{jk}y_k\Big) + b_j,$$

where α and β are (given) positive constants, the weights $\{w_{jk}\}$ represent the connection strength between the k-th and j-th neurons, and b_j represents a bias. Note that most modern neural network architectures are related to stationary solutions of the ODE above. In the present work we restrict our focus to a different connection with dynamical systems (cf. (1.1)), which is more recent, mainly because it has been tested more thoroughly. Also, because we are interested in optimal control, we will primarily focus on the ideas presented in [5], [16], and [54]. Nevertheless, for completeness, we also mention some other related works [22] and [57] from a dynamical systems point of view, [60] for universal maps with memory, [43] for problems in the frequency domain, [24] for a Runge-Kutta based NN, PINNs [12] for PDE-related problems, and SINDy [40] for data-driven model discovery. Also, when ReLU is considered as the activation function, a DNN is a high-dimensional, piecewise linear function. Therefore, some techniques from the Finite Element and the Monte Carlo methods can be used for its analysis, cf. [36].

Motivated by ResNets [33], the authors in [16] relate DNNs of type (1.1) to a recurrence relation obtained when numerically solving a system of ODEs. For instance, for given $f : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$ and $y_0 \in \mathbb{R}$, consider the problem: Find a function y, such that:

$$(3.2) \qquad \begin{aligned} y'(t) &= f(t, y(t)), \quad \text{in } (0, T), \\ y(0) &= y_0. \end{aligned}$$

A solution for this system can be approximated, under mild assumptions on y and f, by the Euler method:

$$y(t+\tau) - y(t) = \int_t^{t+\tau} y'(s)ds = \int_t^{t+\tau} f(s, y(s))ds \approx \tau \cdot f(t, y(t)),$$

where we have (formally) used the fundamental theorem of calculus and a left Riemann-sum approximation. Namely, we can understand the layers of

a DNN as samples from a continuous system that evolves from the input to the output. Here, the first and last layers are special cases due to common upsampling/downsampling techniques.

It is worth mentioning, that in [16, B.2] the authors consider $f = f(y(t), t, \theta)$, where θ represents the parameters of the DNN. Namely, θ is independent of t and therefore their ODE system is limited (essentially) to autonomous systems. Thus, DNNs generated with the method given in [16] are smooth by construction. This property allows one to use well-known results in the theory of dynamical systems, control theory, adaptive ODE solvers, among others. An interesting application where smooth trajectories are desired is when self-intersecting trajectories/surfaces are not allowed as in shape optimization (manifold surfaces), cf. [52, 63]. It is clear that the additional smoothness also limits the usability of the model [21]. Obviously, the architecture of a neural network must match its purpose, i.e. the given data and desired application case.

Besides the networks of type (3.2), the present work also focuses on problems where the system underneath depends on its history in a nonlocal way. The latter is most commonly found in systems with *Hysteresis* or delayed effects. Note that the derivative in (3.2) is a local operator, this follows from its pointwise limit definition. Therefore, based on [3, 4, 5] we consider a fractional derivative based approach. As pointed out in [3, 5], this serves two main purposes: it acts as a global operator (memory effect), and the order of a differential equation is allowed to be less than 1, which reduces the smoothness of the system.

3.2. Stability of continuous fractional-DNN

As mentioned before, ResNet-like architectures can be connected to a classical ODE system, and therefore we can apply the well-known theories to analyze the properties of the DNN [16, 54]. A commonly desired property is the continuous dependence on the data. In the context of machine learning this means that input variables, which are "close" should produce outputs of the DNN which are also "close". Of course, in the context of real-life applications, the notions of distance is not always known, neither is the right dimension, nor the smoothness/regularity for the system underneath. Following [3, 5], we consider a DNN architecture that can be related to a different notion of derivative, the so-called fractional derivative, see Section 2.1, and here we show a stability result for this notion of derivative with respect to the initial data. In order to do so, we consider Ω to be an open, bounded and connected subset of $\mathbb{R}^d$, define $E := \left(L^2(\Omega), \| \cdot \|_\Omega \right)$, and

let $f : \mathrm{Dom}(f) \subseteq [0, \infty) \times E \mapsto E$. To establish the stability of continuous Fractional-DNN, we consider a dynamical system for y. Notice that similar structure holds for the continuous Fractional-DNN (cf. (5.6))

$$(3.3) \qquad \partial_t^\gamma y = f(t, y), \qquad \text{with} \qquad y(0) = y_0,$$

where $y_0 \in E$, and f satisfies the standard assumptions:

(H) $\begin{cases} \text{There exist positive constants } T \text{ and } r \text{ such that } f \text{ restricted to} \\ [0, T] \times B_r(y_0) \text{ is continuous, bounded and Lipschitz with} \\ \text{respect to the second argument.} \end{cases}$

The last hypothesis implies there exists $L > 0$ such that

$$\|f(t, y_1) - f(t, y_2)\|_\Omega \leq L \|y_1 - y_2\|_\Omega \quad \forall t \in [0, T], \text{ and } \forall y_1, y_2 \in B_r(y_0).$$

Let us remark that E can be replaced by any other space with the Radon-Nikodym property but based on the most common loss functions we restrict the analysis to $L^2(\Omega)$. From [4] we have the following result connecting the strong and generalized Caputo derivatives

Lemma 3.1. *Let $\gamma \in (0, 1)$, and $T > 0$. If $y \in W^{1,1}((0, T); E)$ then the following equality holds in the $L^1((0, T]; E)$-sense*

$$(3.4) \qquad \partial_t^\gamma y(t) = D_t^\gamma (y - y(0))(t),$$

for a.e. $t \in (0, T]$, where D_t^γ denotes the Left Riemann-Liouville fractional derivative, cf. [4, Definition 2.2].

Proof. The proof follows from [4, Proposition 2.3], and the fact that every reflexive Banach space has the *Radon-Nikodym property*. □

We write the Left Caputo derivative in the generalized Caputo derivative form (3.4), because several of the well-known results, which hold for standard ODEs, also have their counterparts in the generalized Caputo derivative setting. For instance, the solution operator for a non-autonomous fractional ODE can be represented in terms of a Volterra integral, cf. [20, Theorem 2.1]. By using this integral representation, the next proposition shows the stability of the fractional ODE (3.3) with respect to its initial value, when the solution is smooth enough.

Proposition 3.2. *Given $y_0 \in E$, and f that satisfies (H). If $y_\alpha, y_\beta \in W^{1,1}((0,T);E)$ solve (3.3) with initial conditions $y_{\alpha,0}$ and $y_{\beta,0}$ both in $B_r(y_0)$. Then,*

$$(3.5) \qquad \|y_\alpha - y_\beta\|_{L^1(0,T;E)} \leq C\|y_{\alpha,0} - y_{\beta,0}\|_\Omega,$$

where $C = C(\gamma, T, L) > 0$.

Proof. By Lemma 3.1, and because E is reflexive and therefore has the Radon-Nikodym property, we can recast (3.3) as (3.4), with $y(0) \in \{y_{\alpha,0}, y_{\beta,0}\}$, and represent each solution in terms of a nonlinear Volterra integral, cf. [20, Lemma 2.1]. Namely, if y_α, y_β represent the solutions for (3.3) with corresponding initial conditions $y_{\alpha,0}, y_{\beta,0}$, then for $t \in [0,T]$, and a.e. $x \in \Omega$

$$y_\alpha(x,t) = y_{\alpha,0}(x) + c_\gamma \int_0^t \frac{1}{(t-\tau)^\gamma} f(\tau, y_\alpha(x,\tau))d\tau,$$

$$y_\beta(x,t) = y_{\beta,0}(x) + c_\gamma \int_0^t \frac{1}{(t-\tau)^\gamma} f(\tau, y_\beta(x,\tau))d\tau,$$

where we recall $c_\gamma = \frac{1}{\Gamma(1-\gamma)}$. Then,

$$\|y_\alpha(\cdot,t) - y_\beta(\cdot,t)\|_\Omega$$
$$\leq \|y_{\alpha,0} - y_{\beta,0}\|_\Omega + c_\gamma \int_0^t \frac{1}{(t-\tau)^\gamma} \|f(\tau, y_\alpha) - f(\tau, y_\beta)\|_\Omega \, d\tau$$
$$\leq \|y_{\alpha,0} - y_{\beta,0}\|_\Omega + c_\gamma \int_0^t \frac{1}{(t-\tau)^\gamma} L \|y_\alpha(\cdot,\tau) - y_\beta(\cdot,\tau)\|_\Omega \, d\tau$$

Finally, from Gronwall's inequality in its integral form

$$\|y_\alpha(\cdot,t) - y_\beta(\cdot,t)\|_\Omega \leq \|y_\alpha, - y_{\beta,0}\|_\Omega \exp\left(\frac{L}{\Gamma(1-\gamma)} \int_0^t \frac{1}{(t-\tau)^\gamma} d\tau\right)$$
$$= \|y_{\alpha,0} - y_{\beta,0}\|_\Omega \exp\left(\frac{L}{\Gamma(1-\gamma)} \frac{t^{1-\gamma}}{1-\gamma}\right),$$

and integrating over $(0,T)$ concludes the proof. □

Remark 3.3. It is important to point out that the previous results assume the regularity $W^{1,1}((0,T);E)$, but for most problems in Machine Learning the regularity of solutions is still an open question. Even at the "discrete level" the regularity depends on the data, DNN architecture, optimization

algorithm, loss function, among others factors. Another difficulty is that DNNs can have different number of neurons in each layer, i.e., the space E can change in time.

Finally, and as mentioned before, a DNN with Lipschitz activation functions defines a locally Lipschitzian operator. Later in Section 6, we will explore how the activation function and weights affect locally the gradient of a DNN, and therefore the Lipschitz constant, and we will study the vanishing and exploding gradients problem of various DNNs under the variable$-\tau$ framework which is introduced in Section 5.

4. Network architectures with fixed τ-parameter

Let us begin by stressing that in general the proposed framework with variable τ can be applied to any DNN. We will illustrate our ideas using three representative DNNs. Subsequently, we will describe their strengths and weaknesses.

The first network architecture is ResNet [33]. As described in the previous section (see (3.2)), this network arises after adding an identity map to a standard feedforward network. This leads to connectivity between the adjacent layers. In order to connect all layers and additionally be able to approximate non-smooth functions, we refer to DenseNet [37] and Fractional-DNN [5]. We remark that there also exist other approaches that attempt to induce multilayer connections, e.g. Highway Net [58], AdaNet [18], ResNetPlus [15], etc.

DenseNet is an ad-hoc method that uses the feature maps of all preceding layers as inputs into all subsequent layers. Meanwhile, Fractional-DNN can be viewed as a time-discretization of a fractional in time non-linear ODE of type (3.3), connecting all layers in a mathematically rigorous manner. Both approaches, DenseNet and Fractional-DNN, improve the vanishing gradient effect issue due to the memory effect incorporated. Furthermore, Fractional-DNN allows approximation of non-smooth functions and thus can also potentially help with exploding gradients.

We recall the **ResNet** [33] with equidistant time-steps τ, cf. (1.1). The feature vector $y^{[\ell]} \in \mathbb{R}^{n_\ell}$ in layer $\ell = 1, \ldots, L$ is computed by forward propagation in the following way

$$y^{[\ell]} = P_{\ell-1}^\ell y^{[\ell-1]} + \tau\sigma\left(W^{[\ell-1]}y^{[\ell-1]} + b^{[\ell-1]}\right), \qquad \ell = 1, \ldots, L,$$

where $P_0^1 = 0 \in \mathbb{R}^{n_0 \times n_1}$ and $y^{[0]} = u$. Here, σ is a nonlinear activation function, for instance, ReLU [26], τ is the fixed time-step length and u is

the input data. Notice that, if all the layers are of same size, then $P_{\ell-1}^{\ell}$ equals an identity matrix. In general, $P_{\ell-1}^{\ell}$ will allow layers to have different sizes, i.e.,

$$\dim(P_{\ell-1}^{\ell} y^{[\ell-1]}) = \dim(y^{[\ell]}).$$

While ResNet does introduce connectivity between adjacent layers, we are also interested in fully connected networks, for instance, a DenseNet [37].

In a **DenseNet** the connection through all layers is achieved by the following forward propagation

$$y^{[\ell]} = \sum_{i=0}^{\ell-1} P_i^{\ell} y^{[i]} + \sigma\left(W^{[\ell-1]} y^{[\ell-1]} + b^{[\ell-1]}\right), \qquad \ell = 1, \ldots, L,$$

where $y^{[0]} = u$ is the input data.

Notice that this method does not contain a time step-size like parameter. It is possible to artificially add a parameter τ before the activation function σ. The connection of the resulting expression to a dynamical system remains unclear. Instead of DenseNet, we focus on Fractional-DNN. In addition, to connecting all layers, the Fractional-DNN can be understood as a time-discretization of a dynamical system of type (3.3). Hence, learning the time step-sizes is a meaningful task in this setup.

We recall the **Fractional-DNN**, with equidistant time-steps τ, from [5, 3]. It corresponds to a time-discretization of a system of type (3.3). The forward propagation for the Fractional-DNN is given by

$$y^{[\ell]} = P_{\ell-1}^{\ell} y^{[\ell-1]} - \sum_{j=1}^{\ell-1} a_{\ell-j}(P_j^{\ell} y^{[j]} - P_{j-1}^{\ell} y^{[j-1]})$$
$$+ \tau^{\gamma} \Gamma(2-\gamma) \sigma\left(W^{[\ell-1]} y^{[\ell-1]} + b^{[\ell-1]}\right),$$

where $\ell = 1, \ldots, L$, $y^{[0]} = u$, and P_j^{ℓ} as before. Moreover

$$a_{\ell-j} := (\ell+1-j)^{1-\gamma} - (\ell-j)^{1-\gamma}.$$

Remark 4.1. In Section 5.2 below, we will consider a Fractional-DNN with variable τ, i.e., $\tau^{[\ell]}$ for $\ell = 1, \ldots, L-1$. In this case, the coefficients $a_{\ell-j}$ will depend on $\tau^{[j]}, \ldots, \tau^{[\ell]}$.

We conclude this section by emphasizing that depending on the number of DNN outputs, the last layer may have different size, which can be captured

via

$$y^{[L]} = W^{[L-1]} y^{[L-1]}.$$

For the remainder of the paper, we will assume such a setup for the last layer.

5. Variable-τ framework for DNNs

Instead of a fixed τ, we propose to use a different $\tau^{[\ell]}$ for each layer, which is learned during the training process. This allows us to optimize the "time step-sizes" $\tau^{[\ell]}$, resulting in what can be viewed as an adaptive time-discretization of the ODE tailored to the optimization (learning) problem. The resulting optimization problem is given by (cf. 2.3)

(5.1)
$$\min_{\{W^{[\ell]}\}_{\ell=0}^{L-1}, \{b^{[\ell]}\}_{\ell=0}^{L-2}, \{\tau^{[\ell]}\}_{\ell=0}^{L-2}} J_\lambda \left(\{(y^{[L](i)}, S(u^{(i)}))\}_i, \{W^{[\ell]}\}_\ell, \{b^{[\ell]}\}_\ell, \{\tau^{[\ell]}\}_\ell \right)$$
$$\text{subject to} \quad y^{[L](i)} = \mathcal{F}\left(u^{(i)}; (\{W^{[\ell]}\}_\ell, \{b^{[\ell]}\}_\ell, \{\tau^{[\ell]}\}_\ell) \right) \quad i = 1, \dots, N.$$

Constraints on $\tau^{[\ell]}$, for instance, non-negativity can be easily incorporated. Recall, from (2.4) that J_{λ_1} contains the regularization for the weights $W^{[\ell]}$ and $b^{[\ell]}$. Additional regularization on $\tau^{[\ell]}$ can be easily introduced as

$$J_\lambda := J_{\lambda_1} + \frac{\lambda_2}{2} \sum_{\ell=0}^{L-2} \left(\|\tau^{[\ell]}\|_2^2 + \|\tau^{[\ell]}\|_1 \right),$$

where $\lambda = \lambda(\lambda_1, \lambda_2)$.

We apply the τ-variable framework to the ResNet and the Fractional-DNN discussed above.

5.1. ResNet with variable τ

Consider (5.1) with $\mathcal{F}$ denoting the ResNet with variable τ

(5.2)
$$y^{[\ell]} = P_{\ell-1}^\ell y^{[\ell-1]} + \tau^{[\ell-1]} \sigma(W^{[\ell-1]} y^{[\ell-1]} + b^{[\ell-1]}), \qquad \ell = 1, \dots, L-1,$$
$$y^{[L]} = W^{[L-1]} y^{[L-1]}.$$

For simplicity of notation, we write J instead of J_λ and collect $W^{[\ell]}$, $b^{[\ell]}$, and $\tau^{[\ell]}$ for all ℓ into one vector θ and denote the adjoint variables by ϕ.

Following the approach from [5] and to derive the optimality system, we introduce the Lagrangian functional

$$\mathcal{L}(y, \theta, \phi) = J(\theta) - \sum_{\ell=1}^{L-1} \left\langle y^{[\ell]} - P_{\ell-1}^{\ell} y^{[\ell-1]} - \tau^{[\ell-1]} \sigma(W^{[\ell-1]} y^{[\ell-1]} + b^{[\ell-1]}), \phi^{[\ell]} \right\rangle$$
$$- \left\langle y^{[L]} - W^{[L-1]} y^{[L-1]}, \phi^{[L]} \right\rangle.$$

Setting the variation of $\mathcal{L}$ with respect to ϕ equals zero, we recover the *state equation* (5.2). Similarly, setting the variation of $\mathcal{L}$ with respect to y equals zero, we arrive at the *adjoint system*

$$\phi^{[\ell]} = (P_\ell^{\ell+1})^\top \phi^{[\ell+1]} + \tau^{[\ell]} (W^{[\ell]})^\top \left(\phi^{[\ell+1]} \odot \sigma'(W^{[\ell]} y^{[\ell]} + b^{[\ell]}) \right),$$
$$\ell = L - 2, \dots, 1,$$

$$\phi^{[L-1]} = (W^{[L-1]})^\top \phi^{[L]},$$
$$\phi^{[L]} = \partial_{y^{[L]}} J(\theta) = y^{[L]} - S(u),$$

where the last equality is due to the specific choice of the least-squares loss function. It will be different in case of the cross-entropy softmax, for instance.

Since we will be solving the above optimization problem using a gradient-based method, we also need to evaluate the derivatives with respect to θ:

$$\partial_{W^{[L-1]}} \mathcal{L} = \phi^{[L]} (y^{[L-1]})^\top + \partial_{W^{[L-1]}} J(\theta),$$
$$\partial_{W^{[\ell]}} \mathcal{L} = y^{[\ell]} \left(\phi^{[\ell+1]} \odot \tau^{[\ell]} \sigma'(W^{[\ell]} y^{[\ell]} + b^{[\ell]}) \right)^\top + \partial_{W^{[\ell]}} J(\theta),$$
$$\partial_{b^{[\ell]}} \mathcal{L} = (\phi^{[\ell+1]})^\top \tau^{[\ell]} \sigma' \left(W^{[\ell]} y^{[\ell]} + b^{[\ell]} \right) + \partial_{b^{[\ell]}} J(\theta),$$
$$\partial_{\tau^{[\ell]}} \mathcal{L} = \left\langle \sigma(W^{[\ell]} y^{[\ell]} + b^{[\ell]}), \phi^{[\ell+1]} \right\rangle + \partial_{\tau^{[\ell]}} J(\theta),$$

for $\ell = 0, \dots, L - 2$.

Next, we state the Fractional-DNN [5] but now with variable $\tau^{[\ell]}$. Recall that, in contrast to a ResNet, the Fractional-DNN allows connectivity between all the layers.

5.2. Fractional-DNN with variable τ

Consider a time-discretization $t_0 \leq t_1 \leq \cdots \leq t_L$ with $L \in \mathbb{N}$ and set $I_\ell := (t_\ell, t_{\ell+1}]$ and $\tau^{[\ell]} = t_{\ell+1} - t_\ell$ for $0 \leq \ell \leq L - 1$. Throughout, we will assume that $\tau^{[\ell]} > 0$ to justify division by $\tau^{[\ell]}$.

We generalize the numerical scheme introduced in [45, 44] to a non-equidistant time discretization and obtain the discrete approximation of the left-sided Caputo fractional derivative of order $\gamma \in (0, 1)$. For $0 \le \ell \le L - 1$, we have that:

$$(5.3) \quad \partial_t^\gamma y(x, t_{\ell+1}) = c_\gamma \int_0^{t_{\ell+1}} \frac{\partial_t y(x, t)}{(t_{\ell+1} - t)^\gamma} dt = c_\gamma \sum_{j=0}^\ell \int_{I_j} \frac{\partial_t y(x, t)}{(t_{\ell+1} - t)^\gamma} dt$$

$$= c_\gamma \sum_{j=0}^\ell \frac{y(x, t_{j+1}) - y(x, t_j)}{\tau^{[j]}} \int_{I_j} \frac{1}{(t_{\ell+1} - t)^\gamma} dt + r_\gamma^{\ell+1}$$

where we have used the finite difference approximation. Here $r_\gamma^{\ell+1}$ denotes the remainder from the Taylor formula which can be estimated as described in [49, Section 3.2.1]. After carrying out the integration in (5.3), we arrive at

(5.4)

$$\partial_t^\gamma y(x, t_{\ell+1}) = c_\gamma \sum_{j=0}^\ell \frac{y(x, t_{j+1}) - y(x, t_j)}{\tau^{[j]}} \frac{1}{(1 - \gamma)}$$

$$\cdot \left(\left(\sum_{i=j}^\ell \tau^{[i]} \right)^{1-\gamma} - \left(\sum_{i=j+1}^\ell \tau^{[i]} \right)^{1-\gamma} \right) + r_\gamma^{\ell+1}$$

$$= c_{\gamma-1} \sum_{j=0}^\ell \frac{1}{\tau^{[j]}} \left(\left(\sum_{i=j}^\ell \tau^{[i]} \right)^{1-\gamma} - \left(\sum_{i=j+1}^\ell \tau^{[i]} \right)^{1-\gamma} \right)$$

$$\cdot (y(x, t_{j+1}) - y(x, t_j)) + r_\gamma^{\ell+1}.$$

Analogously, we obtain the approximation of the right-sided Caputo fractional derivative of order $\gamma \in (0, 1)$ for $0 \le \ell \le L - 1$:

$$(5.5) \quad \partial_{T-t}^\gamma y(x, t_\ell) = -c_{\gamma-1} \sum_{j=\ell}^{L-1} \frac{1}{\tau^{[j]}} \left(\left(\sum_{i=\ell}^j \tau^{[i]} \right)^{1-\gamma} - \left(\sum_{i=\ell}^{j-1} \tau^{[i]} \right)^{1-\gamma} \right)$$

$$\cdot (y(x, t_{j+1}) - y(x, t_j)) + r_\gamma^\ell.$$

Before, we apply the above discretization to the Fractional-DNN formulation, we consider two generic nonlinear ODEs of type (3.3) (cf. e.g. [5,

Section 4.1]) with $\gamma \in (0, 1)$:

$$\begin{array}{ll} \partial_t^\gamma y(x, t) = f(t, y(x, t)), & y(x, 0) = y_0, \\ \partial_{T-t}^\gamma y(x, t) = f(t, y(x, t)), & y(x, T) = y_T. \end{array}$$ (5.6)

This also links back to Subsection 3.2, where the stability of the above continuous DNN was discussed. Here, we will move on to formulate the discrete version.

Using the discretizations from (5.4) and (5.5) in (5.6), for $0 \leq \ell \leq L-1$, we arrive at

$$y(\cdot, t_{\ell+1}) = y(\cdot, t_\ell) - \sum_{j=0}^{\ell-1} a_{\ell,j}(y(\cdot, t_{j+1}) - y(\cdot, t_j)) + (\tau^{[\ell]})^\gamma c_{\gamma-1}^{-1} f(t_\ell, y(\cdot, t_\ell)),$$

$$y(\cdot, t_\ell) = y(\cdot, t_{\ell+1}) + \sum_{j=\ell+1}^{L-1} b_{j,\ell}(y(\cdot, t_{j+1}) - y(\cdot, t_j)) + (\tau^{[\ell]})^\gamma c_{\gamma-1}^{-1} f(t_\ell, y(\cdot, t_\ell)),$$

where we denote $y(\cdot, t_\ell) = y(x, t_\ell)$ in the above equations, with

$$a_{\ell,j} := \frac{(\tau^{[\ell]})^\gamma}{\tau^{[j]}} \left(\left(\sum_{i=j}^{\ell} \tau^{[i]} \right)^{1-\gamma} - \left(\sum_{i=j+1}^{\ell} \tau^{[i]} \right)^{1-\gamma} \right),$$

$$b_{j,\ell} := \frac{(\tau^{[\ell]})^\gamma}{\tau^{[j]}} \left(\left(\sum_{i=\ell}^{j} \tau^{[i]} \right)^{1-\gamma} - \left(\sum_{i=\ell}^{j-1} \tau^{[i]} \right)^{1-\gamma} \right).$$

Notice that the equidistant case, $\tau^{[\ell]} = \tau$ for all ℓ, considered throughout the literature, is a special case of the above setting. Additionally, in the equidistant setting, we have $a_{j,\ell} = b_{j,\ell}$, which may not hold in the above generic setting.

After these preparations, we are ready to apply the τ-variable framework to Fractional-DNN. Here, we take into account that the feature vectors $y^{[\ell]}$ may have different sizes across the layers. Thus, as in case of τ-variable ResNet, we introduce projection matrices P_j^ℓ for $j = 0, \ldots, \ell - 1$ and $\ell = 1, \ldots, L - 1$ with $\dim(P_j^\ell y^{[j]}) = \dim(y^{[\ell]})$. The resulting Fractional DNN

with variable τ is

(5.7)
$$
\begin{aligned}
y^{[\ell]} &= P_{\ell-1}^{\ell} y^{[\ell-1]} - \sum_{j=0}^{\ell-2} a_{\ell-1,j} (P_{j+1}^{\ell} y^{[j+1]} - P_j^{\ell} y^{[j]}) \\
&\quad + (\tau^{[\ell-1]})^{\gamma} c_{\gamma-1}^{-1} \sigma(W^{[\ell-1]} y^{[\ell-1]} + b^{[\ell-1]}), \qquad \ell = 1, \ldots, L-1 \\
y^{[L]} &= W^{[L-1]} y^{[L-1]}.
\end{aligned}
$$

Remark 5.1. Before we proceed further, we stress that the τ-variable framework is not merely a scaling of the activation by $\tau^{[\ell]}$. Indeed, in (5.7) the scaling in front of σ is not simply $\tau^{[\ell]}$ but is $(\tau^{\ell-1})^{\gamma} c_{\gamma-1}^{-1}$. Furthermore, $a_{\ell-1,j}$ also contains $\tau^{[j]}, \ldots, \tau^{[\ell-1]}$, which makes the impact of the time step-sizes much more complex than scaling of σ.

As in the ResNet case we next derive the optimality conditions. This requires introducing the Lagrangian formulation as before. In this fractional derivative setting, we observe a subtle issue. It is well-known that there are two approaches to derive the optimality conditions – optimize-then-discretize and discretize-then-optimize [6, 35]. Below, in the τ-variable fractional setting, we observe that the two approaches do not coincide. It is not difficult to see that in the first case, optimize-then-discretize, we obtain the following adjoint equation for $\ell = L-2, \ldots, 1$,

(5.8)
$$
\begin{aligned}
\phi^{[\ell]} &= (P_{\ell}^{\ell+1})^{\top} \phi^{[\ell+1]} + \sum_{j=\ell+1}^{L-2} b_{j,\ell} ((P_{\ell}^{j+1})^{\top} \phi^{[j+1]} - (P_{\ell}^{j})^{\top} \phi^{[j]}) \\
&\quad + (\tau^{[\ell]})^{\gamma} c_{\gamma-1}^{-1} \left[(W^{[\ell]})^{\top} \left(\phi^{[\ell+1]} \odot \sigma'(W^{[\ell]} y^{[\ell]} + b^{[\ell]}) \right) \right], \\
\phi^{[L-1]} &= \left(W^{[L-1]} \right)^{\top} \phi^{[L]}, \\
\phi^{[L]} &= \partial_{y^{[L]}} J(\theta).
\end{aligned}
$$

Next, we derive the adjoint equations for the second approach, i.e., discretize-then-optimize. We begin by introducing the Lagrangian

$$
\begin{aligned}
\mathcal{L}(y, \theta, \phi) =\ & J(\theta) - \sum_{\ell=1}^{L-1} \langle y^{[\ell]} - P_{\ell-1}^{\ell} y^{[\ell-1]} + \sum_{j=0}^{\ell-2} a_{\ell-1,j} (P_{j+1}^{\ell} y^{[j+1]} - P_j^{\ell} y^{[j]}) \\
&\quad - (\tau^{[\ell-1]})^{\gamma} c_{\gamma-1}^{-1} \sigma(W^{[\ell-1]} y^{[\ell-1]} + b^{[\ell-1]}), \phi^{[\ell]} \rangle \\
&\quad - \left\langle y^{[L]} - W^{[L-1]} y^{[L-1]}, \phi^{[L]} \right\rangle.
\end{aligned}
$$

Setting the variation of $\mathcal{L}$ with respect to ϕ equal zero, we obtain the state equation (5.7). To derive the adjoint equation, we calculate the variation of $\mathcal{L}$ with respect to $y^{[\ell]}$ for every $\ell = 1, \ldots, L$. A detailed calculation can be found in Appendix A.1. Setting this variation equal to zero, we arrive at the following adjoint system for $\ell = L - 2, \ldots, 1$,

(5.9)

$$
\phi^{[\ell]} = (1 - a_{\ell,\ell-1})(P_\ell^{\ell+1})^\top \phi^{[\ell+1]} + \sum_{j=\ell+2}^{L-1} (a_{j-1,\ell} - a_{j-1,\ell-1})(P_\ell^j)^\top \phi^{[j]}
$$
$$
+ (\tau^{[\ell]})^\gamma c_{\gamma-1}^{-1} \left[(W^{[\ell]})^\top \left(\phi^{[\ell+1]} \odot \sigma'(W^{[\ell]} y^{[\ell]} + b^{[\ell]}) \right) \right],
$$
$$
\phi^{[L-1]} = \left(W^{[L-1]} \right)^\top \phi^{[L]},
$$
$$
\phi^{[L]} = \partial_{y^{[L]}} J(\theta).
$$

Below, we collect all summands that contain factors $b_{j,\ell}$ in (5.8) on the left side, and all summands that contain factors $a_{j,\ell}$ in (5.9) on the right side. We see that the two adjoint equations given in (5.8) and (5.9) differ in the following term

$$
\sum_{j=\ell+1}^{L-2} b_{j,\ell}((P_\ell^{j+1})^\top \phi^{[j+1]} - (P_\ell^j)^\top \phi^{[j]})
$$
$$
\neq \sum_{j=\ell+1}^{L-2} a_{j,\ell}(P_\ell^{j+1})^\top \phi^{[j+1]} - \sum_{j=\ell+1}^{L-1} a_{j-1,\ell-1}(P_\ell^j)^\top \phi^{[j]}
$$

In our computations, we have implemented the discretize-then-optimize approach, i.e. (5.9). Finally, we compute the derivative with respect to θ:

$$
\partial_{W^{[L-1]}} \mathcal{L} = \phi^{[L]}(y^{[L-1]})^\top + \partial_{W^{[L-1]}} J(\theta),
$$
$$
\partial_{W^{[\ell]}} \mathcal{L} = y^{[\ell]} \left(\phi^{[\ell+1]} \odot (\tau^{[\ell]})^\gamma c_{\gamma-1}^{-1} \sigma'(W^{[\ell]} y^{[\ell]} + b^{[\ell]}) \right)^\top + \partial_{W^{[\ell]}} J(\theta),
$$
$$
\partial_{b^{[\ell]}} \mathcal{L} = (\phi^{[\ell+1]})^\top (\tau^{[\ell]})^\gamma c_{\gamma-1}^{-1} \sigma' \left(W^{[\ell]} y^{[\ell]} + b^{[\ell]} \right) + \partial_{b^{[\ell]}} J(\theta),
$$
$$
\partial_{\tau^{[\ell]}} \mathcal{L} = -\sum_{k=\ell}^{L-2} \sum_{j=0}^{\min\{k-1,\ell\}} \partial_{\tau^{[\ell]}}(a_{k,j}) \left\langle P_{j+1}^{k+1} y^{[j+1]} - P_j^{k+1} y^{[j]}, \phi^{[k+1]} \right\rangle
$$
$$
+ \left\langle \gamma(\tau^{[\ell]})^{\gamma-1} c_{\gamma-1}^{-1} \sigma(W^{[\ell]} y^{[\ell]} + b^{[\ell]}), \phi^{[\ell+1]} \right\rangle + \partial_{\tau^{[\ell]}} J(\theta).
$$

for $\ell = 0, \ldots, L - 2$. Details on the computation of $\partial_{\tau^{[\ell]}} \mathcal{L}$ can be found in Appendix A.2. Next, we examine the impact of variable τ onto the stability of networks such as ResNets and Fractional-DNNs.

6. Vanishing and exploding gradients

It is well known that optimization problems with DNN constraints can suffer from vanishing and exploding gradients, see e.g. [9, 25]. In this section, we analyze the structure of the derivatives for several network architectures such as feedforward network, ResNet, DenseNet, Fractional-DNN, and the consequences of application of τ-variable framework on these networks. We will identify various conditions to help overcome the aforementioned challenges.

For simplicity of the notation, we define the abbreviation $a^{[\ell]} :=$ $\sigma(W^{[\ell]} y^{[\ell]} + b^{[\ell]})$ and omit the projection matrices $P_{\ell-1}^{\ell}$, i.e. $n_\ell = n$ for all layers ℓ. While the following result may not be new for the standard case with $\tau^{[\ell]} = \tau \in \mathbb{R}$ for all ℓ, but to the best of our knowledge, this is new for variable $\tau^{[\ell]}$.

Theorem 6.1 (Feedforward Network and ResNet). *Consider the feedforward network and ResNet with τ-variable framework*

$$y^{[\ell]} = \tau^{[\ell-1]} a^{[\ell-1]}, \qquad\qquad \ell = 1, \ldots, L-1,$$
$$y^{[\ell]} = y^{[\ell-1]} + \tau^{[\ell-1]} a^{[\ell-1]}, \qquad \ell = 1, \ldots, L-1.$$

Let $\theta^{[k]} = (W^{[k]}(:), b^{[k]}, \tau^{[k]})^{\top}$ be the parameters associated with layer k for $k = 0, \ldots, L-2$. Then the respective derivatives take the form

$$(6.1) \qquad d_{\theta^{[j]}} y^{[\ell]} = \prod_{i=\ell-1}^{j+1} \left(\tau^{[i]} d_{y^{[i]}} a^{[i]} \right) \partial_{\theta^{[j]}} (\tau^{[j]} a^{[j]}),$$

$$(6.2) \qquad d_{\theta^{[j]}} y^{[\ell]} = \prod_{i=\ell-1}^{j+1} \left(\mathbb{I} + \tau^{[i]} d_{y^{[i]}} a^{[i]} \right) \partial_{\theta^{[j]}} (\tau^{[j]} a^{[j]}),$$

for all $\ell = 1, \ldots, L-1$ and $j = 0, \ldots, \ell-1$.

Proof. For the feedforward neural network we can compute with chain rule

$$d_{\theta^{[j]}} y^{[\ell]} = d_{\theta^{[j]}} \tau^{[\ell-1]} a^{[\ell-1]}$$
$$= \tau^{[\ell-1]} d_{y^{[\ell-1]}} a^{[\ell-1]} \cdot d_{\theta^{[j]}} y^{[\ell-1]}$$

$$= \tau^{[\ell-1]} \mathrm{d}_{y^{[\ell-1]}} a^{[\ell-1]} \cdot \mathrm{d}_{\theta^{[j]}} \tau^{[\ell-2]} a^{[\ell-2]}.$$

where $\cdot$ denotes the standard matrix multiplication. By iterating we arrive at

$$\mathrm{d}_{\theta^{[j]}} y^{[\ell]} = \prod_{i=\ell-1}^{j+1} \left(\tau^{[i]} \mathrm{d}_{y^{[i]}} a^{[i]} \right) \mathrm{d}_{\theta^{[j]}} y^{[j+1]} = \prod_{i=\ell-1}^{j+1} \left(\tau^{[i]} \mathrm{d}_{y^{[i]}} a^{[i]} \right) \partial_{\theta^{[j]}} (\tau^{[j]} a^{[j]}).$$

Similarly, for the ResNet, we obtain

$$\begin{aligned}
\mathrm{d}_{\theta^{[j]}} y^{[\ell]} &= \mathrm{d}_{\theta^{[j]}} (y^{[\ell-1]} + \tau^{[\ell-1]} a^{[\ell-1]}) \\
&= \mathrm{d}_{\theta^{[j]}} y^{[\ell-1]} + \tau^{[\ell-1]} \mathrm{d}_{y^{[\ell-1]}} a^{[\ell-1]} \cdot \mathrm{d}_{\theta^{[j]}} y^{[\ell-1]} \\
&= (\mathbb{I} + \tau^{[\ell-1]} \mathrm{d}_{y^{[\ell-1]}} a^{[\ell-1]}) \mathrm{d}_{\theta^{[j]}} y^{[\ell-1]},
\end{aligned}$$

where $\mathbb{I} \in \mathbb{R}^{n \times n}$ denotes the identity matrix, with $n = n_\ell$ constant throughout all layers ℓ. By iterating we arrive at

$$\begin{aligned}
\mathrm{d}_{\theta^{[j]}} y^{[\ell]} &= \prod_{i=\ell-1}^{j+1} \left(\mathbb{I} + \tau^{[i]} \mathrm{d}_{y^{[i]}} a^{[i]} \right) \mathrm{d}_{\theta^{[j]}} y^{[j+1]} \\
&= \prod_{i=\ell-1}^{j+1} \left(\mathbb{I} + \tau^{[i]} \mathrm{d}_{y^{[i]}} a^{[i]} \right) \partial_{\theta^{[j]}} (\tau^{[j]} a^{[j]}),
\end{aligned}$$

where we use that $\mathrm{d}_{\theta^{[j]}} y^{[j]} = 0$. This concludes the proof. $\qquad \square$

Remark 6.2. Since σ is applied componentwise, special caution needs to be exercised when deriving $\mathrm{d}_{y^{[i]}} a^{[i]}$. Let $r_j^{[i]}$ be the jth row of $W^{[i]} y^{[i]} + b^{[i]}$, namely $\sum_{m=1}^n W_{j,m}^{[i]} y_m^{[i]} + b_j^{[i]}$ for $j \in \{1, \dots, n\}$. Then, with a slight abuse of notation, it holds

$$\begin{aligned}
\mathrm{d}_{y^{[i]}} a^{[i]} &= \mathrm{d}_{y^{[i]}} \sigma(W^{[i]} y^{[i]} + b^{[i]}) \\
&= \mathrm{d}_{y^{[i]}} \left(\sigma(r_j^{[i]}) \right)_{j=1}^n \\
&= \mathrm{diag}(\sigma'(r_1^{[i]}), \dots, \sigma'(r_n^{[i]})) \cdot W^{[i]},
\end{aligned}$$

where $\sigma'(r_j^{[i]})$ is the one-dimensional derivative of σ at $r_j^{[i]}$. Furthermore, for the partial derivative $\partial_{\theta^{[j]}} (\tau^{[j]} a^{[j]})$, we recall $\theta^{[j]} = (W^{[j]}(:), b^{[j]}, \tau^{[j]})^\top \in \mathbb{R}^N$,

with $N = n^2 + n + 1$ and the fact that $a^{[j]}$ depends on $W^{[j]}$ and $b^{[j]}$, but not $\tau^{[j]}$. Consequently, we see

$$\partial_{\theta^{[j]}}(\tau^{[j]}a^{[j]}) = \begin{pmatrix} \tau^{[j]}\partial_{W^{[j]}(:)}a^{[j]} & \tau^{[j]}\partial_{b^{[j]}}a^{[j]} & a^{[j]} \end{pmatrix} \in \mathbb{R}^{n \times N}.$$

As pointed out above, the standard feedforward neural network, where $\tau^{[\ell]} = 1$ for all ℓ, can suffer from vanishing and exploding gradients, which can be a challenge for optimization with deep networks [9, 25]. Consider the structure of the derivatives in (6.1) with $\tau^{[\ell]} = 1$ for all ℓ, e.g. for the final hidden layer with $\ell = L - 1$,

$$d_{\theta^{[j]}}y^{[L-1]} = \prod_{i=L-2}^{j+1} \left(d_{y^{[i]}}a^{[i]} \right) \partial_{\theta^{[j]}}a^{[j]}.$$

Especially in the one-dimensional case, it is obvious that if the partial derivatives $d_{y^{[i]}}a^{[i]}$ are smaller than 1, the product will tend to 0 as the number of layers L increases, which leads to vanishing gradients. On the other hand, if the partial derivatives $d_{y^{[i]}}a^{[i]}$ are larger than 1, the product will tend to ∞ as the number of layers L increases, which leads to exploding gradients. The feedforward neural network with variable τ, can potentially help overcome both problems, since now we have flexibility with respect to $\tau^{[\ell]}$. But one needs to be careful as if the gradient components are really small, then $\tau^{[\ell]}$ needs to be really large to compensate, which could lead to ill-conditioning issues.

A more appropriate approach is the standard ResNet with $\tau^{[\ell]} = \tau \in \mathbb{R}$ for all ℓ. It is known to be stable with respect to vanishing gradients. Recalling the gradient from (6.2)

$$d_{\theta^{[j]}}y^{[L-1]} = \prod_{i=L-2}^{j+1} \left(\mathbb{I} + \tau^{[i]}d_{y^{[i]}}a^{[i]} \right) \partial_{\theta^{[j]}}(\tau^{[j]}a^{[j]}),$$

it becomes clear that this stability is achieved by the added identity $\mathbb{I}$ in every part of the product. Hence, even if the Jacobians $d_{y^{[i]}}a^{[i]}$ vanish, the product still contains the identity matrices. This advantage carries over to the τ-variable framework.

Furthermore, the introduction of $\tau^{[\ell]}$ in ResNet allows us to tackle the exploding gradients problem. This property has also been discussed, using probabilistic bounds, in [31]. Our approach is deterministic. The standard ResNet architecture does not have this property. Appropriate small $\tau^{[\ell]}$ can

prevent the product from exploding with growing number of layers. However, choosing $\tau^{[\ell]}$ too small may lead to vanishing gradient problem again, as we will illustrate in the following simple example. Recall that, we do not tune $\tau^{[\ell]}$ by hand, but let the optimization find it.

Example 6.3. Consider the ResNet architecture with variable τ in one dimenstion, i.e. one node per layer. We have

$$
\begin{aligned}
\mathrm{d}_{\theta^{[1]}} y^{[2]} &= \partial_{\theta^{[1]}} (\tau^{[1]} a^{[1]}), \\
\mathrm{d}_{\theta^{[0]}} y^{[2]} &= (1 + \tau^{[1]} \mathrm{d}_{y^{[1]}} a^{[1]}) \, \partial_{\theta^{[0]}} (\tau^{[0]} a^{[0]}).
\end{aligned}
$$

Assume that $\mathrm{d}_{y^{[1]}} a^{[1]}$ is large, so that it leads to a large $\mathrm{d}_{\theta^{[0]}} y^{[2]}$. This problem can be overcome, if $\tau^{[1]}$ attains a small value. However, this may lead to $\mathrm{d}_{\theta^{[1]}} y^{[2]}$ being accordingly small in its first two components, i.e. the derivatives by the weights and biases. Consequently, fixing one potential exploding gradient problem, can cause another gradient to vanish. However, as emphasized earlier, we do not tune $\tau^{[\ell]}$ by hand, but let the optimization find optimal values.

We also analyze the respective derivatives in the DenseNet architecture with variable τ. Finding a closed form for $\mathrm{d}_{\theta^{[j]}} y^{[\ell]}$ is not so easy for this network architecture, but we can derive a recursive relation in terms of lower order terms.

Theorem 6.4. *Consider the DenseNet with τ-variable framework*

$$
y^{[\ell]} = \sum_{k=0}^{\ell-1} y^{[k]} + \tau^{[\ell-1]} a^{[\ell-1]}, \qquad \ell = 1, \dots, L - 1.
$$

Then the derivatives for $i = \ell, \dots, j + 2$ can be recursively written as

$$
\begin{aligned}
\mathrm{d}_{\theta^{[j]}} y^{[i]} &= \mathrm{d}_{\theta^{[j]}} \sum_{k=j+1}^{i-2} y^{[k]} + (\mathbb{I} + \tau^{[i-1]} \mathrm{d}_{y^{[i-1]}} a^{[i-1]}) \, \mathrm{d}_{\theta^{[j]}} y^{[i-1]}, \\
\mathrm{d}_{\theta^{[j]}} y^{[j+1]} &= \partial_{\theta^{[j]}} (\tau^{[j]} a^{[j]}).
\end{aligned}
$$

Proof. For $i = \ell, \dots, j + 2$ we employ the chain rule of differentiation and use that $\mathrm{d}_{\theta^{[j]}} y^{[k]} = 0$ for $k < j + 1$ to arrive at

$$
\mathrm{d}_{\theta^{[j]}} y^{[i]} = \mathrm{d}_{\theta^{[j]}} \left(\sum_{k=0}^{i-1} y^{[k]} + \tau^{[i-1]} a^{[i-1]} \right)
$$

$$= \mathrm{d}_{\theta^{[j]}} \sum_{k=j+1}^{i-2} y^{[k]} + \mathrm{d}_{\theta^{[j]}} y^{[i-1]} + \tau^{[i-1]} \mathrm{d}_{\theta^{[j]}} a^{[i-1]}$$

$$= \mathrm{d}_{\theta^{[j]}} \sum_{k=j+1}^{i-2} y^{[k]} + (\mathbb{I} + \tau^{[i-1]} \mathrm{d}_{y^{[i-1]}} a^{[i-1]}) \, \mathrm{d}_{\theta^{[j]}} y^{[i-1]}.$$

The case $i = j+1$ is special, since the chain rule does not need to be applied here. It simply holds

$$\mathrm{d}_{\theta^{[j]}} y^{[j+1]} = \mathrm{d}_{\theta^{[j]}} \left(\sum_{k=0}^{j} y^{[k]} + \tau^{[j]} a^{[j]} \right) = \partial_{\theta^{[j]}}(\tau^{[j]} a^{[j]}),$$

because $k < j + 1$. The proof is complete. $\qquad\square$

Remark 6.5. In Theorem 6.4 we can successively insert the expressions for the lower order terms in the higher order terms, so that finally $\mathrm{d}_{\theta^{[j]}} y^{[\ell]}$ depends only on $\partial_{\theta^{[j]}}(\tau^{[j]} a^{[j]})$ and $\mathrm{d}_{y^{[i]}} a^{[i]}$ for $i = j+1, \ldots, \ell-1$. Furthermore, we see that every next lower order term enters with a factor

$$(\mathbb{I} + \tau^{[i]} \mathrm{d}_{y^{[i]}} a^{[i]}),$$

so that one can overcome the vanishing gradients problem in a DenseNet (both fixed and variable τ cases). To discuss the exploding gradients problem we consider for example the derivative $\mathrm{d}_{\theta^{[j]}} y^{[L-1]}$, where one summand will be $\prod_{i=j+1}^{L-2} \left(\tau^{[i]} \mathrm{d}_{y^{[i]}} a^{[i]} \right) \partial_{\theta^{[j]}}(\tau^{[j]} a^{[j]})$. In the standard one-dimensional DenseNet architecture with $\tau^{[\ell]} = 1$ for all ℓ, we see that the above product tends to ∞ with a growing number of layers L if $\mathrm{d}_{y^{[i]}} a^{[i]} > 1$ for all i. The τ-variable architecture can help deal with this problem, see also Example 6.8.

Similarly to the above cases, we can express derivatives of Fractional-DNN architecture, with variable τ, in terms of lower order terms.

Theorem 6.6. *Consider the Fractional-DNN with τ-variable framework*

$$y^{[\ell]} = y^{[\ell-1]} - \sum_{k=0}^{\ell-2} a_{\ell,k}(y^{[k+1]} - y^{[k]}) + (\tau^{[\ell-1]})^{\gamma} c_{\gamma-1}^{-1} a^{[\ell-1]}, \quad \ell = 1, \ldots, L-1.$$

Then the derivatives for $i = \ell, \ldots, j + 2$ can be recursively written as

$$\mathrm{d}_{\theta^{[j]}} y^{[i]} = \mathrm{d}_{\theta^{[j]}} \sum_{k=j+1}^{i-2} (a_{i,k} - a_{i,k-1}) y^{[k]}$$

$$+ \left((1 - a_{i,i-2})\mathbb{I} + (\tau^{[i-1]})^\gamma c_{\gamma-1}^{-1} \, \mathrm{d}_{y^{[i-1]}} a^{[i-1]} \right) \mathrm{d}_{\theta^{[j]}} y^{[i-1]},$$

$$\mathrm{d}_{\theta^{[j]}} y^{[j+1]} = c_{\gamma-1}^{-1} \, \partial_{\theta^{[j]}} ((\tau^{[j]})^\gamma a^{[j]}).$$

Proof. First of all, we rewrite the forward propagation for $\ell = 1, \ldots, L - 1$ in Fractional-DNN

$$y^{[\ell]} = y^{[\ell-1]} - \sum_{k=0}^{\ell-2} a_{\ell,k}(y^{[k+1]} - y^{[k]}) + (\tau^{[\ell-1]})^\gamma c_{\gamma-1}^{-1} a^{[\ell-1]}$$

$$= a_{\ell,0} y^{[0]} + \sum_{k=1}^{\ell-2} (a_{\ell,k} - a_{\ell,k-1}) y^{[k]} + (1 - a_{\ell,\ell-2}) y^{[\ell-1]}$$

$$+ (\tau^{[\ell-1]})^\gamma c_{\gamma-1}^{-1} a^{[\ell-1]}.$$

Then for $i = \ell, \ldots, j + 2$ we use chain rule and $\mathrm{d}_{\theta^{[j]}} y^{[k]} = 0$ for $k < j + 1$ to obtain

$$\mathrm{d}_{\theta^{[j]}} y^{[i]} = \mathrm{d}_{\theta^{[j]}} \left(a_{i,0} y^{[0]} + \sum_{k=1}^{i-2} (a_{i,k} - a_{i,k-1}) y^{[k]} \right.$$

$$\left. + (1 - a_{i,i-2}) y^{[i-1]} + (\tau^{[i-1]})^\gamma c_{\gamma-1}^{-1} a^{[i-1]} \right)$$

$$= \mathrm{d}_{\theta^{[j]}} \sum_{k=j+1}^{i-2} (a_{i,k} - a_{i,k-1}) y^{[k]}$$

$$+ \left((1 - a_{i,i-2})\mathbb{I} + (\tau^{[i-1]})^\gamma c_{\gamma-1}^{-1} \, \mathrm{d}_{y^{[i-1]}} a^{[i-1]} \right) \mathrm{d}_{\theta^{[j]}} y^{[i-1]}.$$

Finally, for $i = j + 1$, we exploit again $\mathrm{d}_{\theta^{[j]}} y^{[k]} = 0$ for $k < j + 1$, and derive

$$\mathrm{d}_{\theta^{[j]}} y^{[j+1]} = \mathrm{d}_{\theta^{[j]}} \left(a_{j+1,0} y^{[0]} + \sum_{k=1}^{j-1} (a_{j+1,k} - a_{j+1,k-1}) y^{[k]} \right.$$

$$\left. + (1 - a_{j+1,j-1}) y^{[j]} + (\tau^{[j]})^\gamma c_{\gamma-1}^{-1} a^{[j]} \right)$$

$$= c_{\gamma-1}^{-1} \, \partial_{\theta^{[j]}} ((\tau^{[j]})^\gamma a^{[j]}).$$

This completes the proof.   □

Remark 6.7. Again, the lower order term representations can be successively inserted into the higher order terms until we arrive at $\mathrm{d}_{\theta^{[j]}} y^{[\ell]}$ depending only on $\partial_{\theta^{[j]}} ((\tau^{[j]})^\gamma a^{[j]})$ and $\mathrm{d}_{y^{[i]}} a^{[i]}$ for $i = j + 1, \ldots, \ell - 1$. Here, the

next lower order term enters with a factor

$$\left((1 - a_{i,i-2})\mathbb{I} + (\tau^{[i-1]})^\gamma c_{\gamma-1}^{-1} \, \mathrm{d}_{y^{[i-1]}} a^{[i-1]}\right),$$

which allows us to overcome the vanishing gradient problem in Fractional-DNNs. Furthermore, the multiplication by $(\tau^{[i-1]})^\gamma$ in this factor can help us to deal with exploding gradients. This is similar to ResNet with variable τ.

Example 6.8. To get an idea of how different network architectures influence the derivatives, the derivative $\mathrm{d}_{\theta^{[0]}} y^{[3]}$ is displayed here for the four different options that have been considered in this section, i.e., feedforward neural network, ResNet, DenseNet and Fractional DNN:

$$\mathrm{d}_{\theta^{[0]}} y^{[3]} = \tau^{[2]} \mathrm{d}_{y^{[2]}} a^{[2]} \cdot \tau^{[1]} \mathrm{d}_{y^{[1]}} a^{[1]} \cdot \partial_{\theta^{[0]}} (\tau^{[0]} a^{[0]}),$$

$$\mathrm{d}_{\theta^{[0]}} y^{[3]} = \left(\mathbb{I} + \tau^{[1]} \mathrm{d}_{y^{[1]}} a^{[1]} + \tau^{[2]} \mathrm{d}_{y^{[2]}} a^{[2]} + \tau^{[1]} \tau^{[2]} \mathrm{d}_{y^{[2]}} a^{[2]} \cdot \mathrm{d}_{y^{[1]}} a^{[1]}\right)$$
$$\cdot \partial_{\theta^{[0]}} (\tau^{[0]} a^{[0]}),$$

$$\mathrm{d}_{\theta^{[0]}} y^{[3]} = \left(2\mathbb{I} + \tau^{[1]} \mathrm{d}_{y^{[1]}} a^{[1]} + \tau^{[2]} \mathrm{d}_{y^{[2]}} a^{[2]} + \tau^{[1]} \tau^{[2]} \mathrm{d}_{y^{[2]}} a^{[2]} \cdot \mathrm{d}_{y^{[1]}} a^{[1]}\right)$$
$$\cdot \partial_{\theta^{[0]}} (\tau^{[0]} a^{[0]}),$$

$$\mathrm{d}_{\theta^{[0]}} y^{[3]} = \left((1 - a_{2,0} - a_{3,0} + a_{2,0} a_{3,1})\mathbb{I} + (1 - a_{3,1})(\tau^{[1]})^\gamma c_{\gamma-1}^{-1} \mathrm{d}_{y^{[1]}} a^{[1]} \right.$$
$$\left. + (1 - a_{2,0})(\tau^{[2]})^\gamma c_{\gamma-1}^{-1} \mathrm{d}_{y^{[2]}} a^{[2]} + (\tau^{[1]})^\gamma (\tau^{[2]})^\gamma c_{\gamma-1}^{-2} \mathrm{d}_{y^{[2]}} a^{[2]} \cdot \mathrm{d}_{y^{[1]}} a^{[1]}\right)$$
$$\cdot c_{\gamma-1}^{-1} \partial_{\theta^{[0]}} ((\tau^{[0]})^\gamma a^{[0]}).$$

In conclusion, ResNet, DenseNet and Fractional-DNN have a visible additive structure in the derivatives, which helps with the vanishing gradients problem. Furthermore, the parameters $\tau^{[\ell]}$ can help overcome both vanishing and exploding gradients.

7. Numerical results

In this section, we apply the τ-variable framework to a ResNet and a Fractional DNN with and without bias ordering (2.5). A thorough comparison is carried out in the context of an ill-posed 3D parametrized Maxwell's equation with Gauss's law. This problem is ill-posed because the standard Nédélec finite element is only curl conforming and cannot directly impose the Gauss's law. In all the cases, we apply the smoothed version of standard

$\text{ReLU}(y) = \max\{0, y\}$ as the activation function

$$\text{smoothReLU}(y) = \begin{cases} \max\{0, y\}, & \text{if } |y| > \eta \\ \frac{1}{4\eta}y^2 + 0.5y + 0.25\eta, & \text{if } y \in [-\eta, \eta]. \end{cases}$$

We have found that $\eta = 10^{-4}$ is a robust choice for the examples under consideration. Notice, that one can also use other activation functions which can differ from layer to layer.

7.1. Maxwell's equations

Our findings suggests that the τ-variable framework outperforms the standard approach (with fixed τ) for deeper networks, see Figure 1. This is expected, since the effect of variable $\tau^{[\ell]}$ will be more prominent when more layers (and consequently more time-step parameters $\tau^{[\ell]}$) are present. On the other hand, for shallow networks, the τ-variable framework provides comparatively less improvements, see Figure 3(c). Nevertheless, the τ-variable framework applied to ResNet yields error improvements compared to a standard ResNet for model extrapolation, see Figure 4. These results are also comparable to the approximation obtained with the finite element method (FEM) with the lowest order Nédélec space, cf. Figure 3(d).

Consider the Maxwell-Dirac equations. Our goal is to learn $\boldsymbol{u} : \Omega \subset \mathbb{R}^3 \mapsto \mathbb{R}^3$ that satisfies

$$\begin{aligned} \text{curl}\left(\boldsymbol{\mu}^{-1}\text{curl}\boldsymbol{u}\right) &= \boldsymbol{f} && \text{in } \Omega, \\ \text{div}(\boldsymbol{\varepsilon}\boldsymbol{u}) &= \rho && \text{in } \Omega, \\ \boldsymbol{u} \times \boldsymbol{n} &= \boldsymbol{g} && \text{on } \partial\Omega, \end{aligned}$$

(7.1)

where $\boldsymbol{\mu}$ and $\boldsymbol{\varepsilon}$ are positive definite symmetric tensors in $L^\infty(\Omega)^3$, $\boldsymbol{f} \in L^2(\Omega)^3$, $\rho \in L^2(\Omega)$ and $\boldsymbol{g} \in H_\parallel^{-\frac{1}{2}}(\text{div}_\Gamma; \partial\Omega)$. This problem is particularly difficult at the discrete level due to its divergence-related constraints and requires rather tailored algorithms to deal with it, see for instance [17]. Therefore, an interesting question is to approximate the map:

$$(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x}), \boldsymbol{\mu}(\boldsymbol{x}), \rho(\boldsymbol{x})) \mapsto \boldsymbol{u}(\boldsymbol{x}),$$

that can lead to a reasonable and noise-robust approximation of the solution $\boldsymbol{u}$ to (7.1), note that we ignore the boundary data $\boldsymbol{g}$. This approach is similar to a surrogate model where its output could be used as an initial guess by an iterative method like the domain decomposition method or in the reduced

basis method [61]. Nevertheless, those problems are beyond the scope of the present paper, and it will be studied in future works.

This learning problem is challenging because the solutions to (7.1) can have discontinuities, while most neural networks, except for the ones with the Heaviside activation function, lead to continuous approximations. Also, it is still not clear how to incorporate the geometry (domain Ω) of the problem in a meaningful way. Thus, we consider an example with a known smooth solution and we compare it with an approximation obtained by various DNNs and by the lowest order Nédélec space of the first kind, cf. [48], denoted by $\mathcal{N}_0(\Omega)$. Here, we consider the basis proposed in [27]. In order to do that, let us consider $\boldsymbol{\varepsilon} = \mathbb{I}_{3\times3}$, and for a smooth $\varphi : \Omega \mapsto \mathbb{R}^+$ we define $\boldsymbol{\mu}^{-1}(\boldsymbol{x}) = \varphi(\boldsymbol{x})\mathbb{I}_{3\times3}$, then $\operatorname{curl}\left(\boldsymbol{\mu}^{-1}\operatorname{curl}\boldsymbol{u}\right) = \nabla\varphi \times \operatorname{curl}\boldsymbol{u} + \varphi\operatorname{curl}(\operatorname{curl}\boldsymbol{u})$.

Thus, if we consider

$$(7.2) \qquad \boldsymbol{u} : \Omega \mapsto \mathbb{R}^3, \quad (x_1, x_2, x_3) \mapsto I_1(r(x_1, x_2, x_3))\boldsymbol{e}_\theta,$$

$$\varphi : \Omega \mapsto \mathbb{R}, \quad (x_1, x_2, x_3) \mapsto \frac{1}{2}(x_1^2 + x_2^2 + 1),$$

where Ω is the cylinder $\{(x_1, x_2, x_3) \in \mathbb{R}^3 : x_1^2 + x_2^2 \leq 1 \text{ and } x_3 \in [0,1]\}$, I_ν is the modified Bessel functions of the first kind of order ν, $r(x_1, x_2, x_3) = \sqrt{x_1^2 + x_2^2}$, and $\boldsymbol{e}_\theta(x_1, x_2, x_3) = (x_1^2 + x_2^2)^{-\frac{1}{2}}(-x_2, x_1, 0)$, we obtain:

$$\operatorname{curl}\boldsymbol{u} = I_0(r)\boldsymbol{e}_z, \quad \operatorname{curl}(\operatorname{curl}\boldsymbol{u}) = -\boldsymbol{u}, \quad \operatorname{div}\boldsymbol{u} = 0, \text{ and}$$

$$\operatorname{curl}\left(\boldsymbol{\mu}^{-1}\operatorname{curl}\boldsymbol{u}\right) = -rI_0(r)\boldsymbol{e}_\theta - \varphi\boldsymbol{u} =: \boldsymbol{f}.$$

Because $\boldsymbol{u}$ is divergence free, we consider the reduced map $(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x}), \boldsymbol{\varphi}(\boldsymbol{x})) \mapsto \boldsymbol{u}(\boldsymbol{x})$, where $\boldsymbol{x} = (x_1, x_2, x_3)$, and $\boldsymbol{u}(\boldsymbol{x}) = (\boldsymbol{u}_1(\boldsymbol{x}), \boldsymbol{u}_2(\boldsymbol{x}), \boldsymbol{u}_3(\boldsymbol{x}))$. In order to generate the input/output data for the DNNs, we consider points $\{\boldsymbol{x}_i\}_{i=1}^N \subset \Omega$ randomly chosen from Ω obtained with Matlab's function unifrnd along with philox as the random number algorithm. Here, we set $N = 12{,}000$. Then, $\{(\boldsymbol{x}_i, f(\boldsymbol{x}_i), \varphi(\boldsymbol{x}_i))\}_{i=1}^N$ and $\{\boldsymbol{u}(\boldsymbol{x}_i)\}_{i=1}^N$ can be utilized as input/output data. We are now ready to train and compare several DNNs.

Neural network size & network reduction

The bigger the network that we use for training, the bigger the computational time and the memory requirements. We employ the τ-variable framework and start with a ResNet architecture with 5 hidden layers with 10 nodes each. As target functional we implement the mean squared error with

no regularization, i.e. $\lambda_1 = \lambda_2 = 0$. Additionally, we consider bias ordering (B.O.) with a fixed Moreau-Yosida parameter $\beta = 10$. After 1000 steepest descent steps we observe the following result: The relative error in the Euclidean norm on the test set is 0.07, and we see $\tau^{[1]}, \tau^{[3]}$ and $\tau^{[4]}$ are approximately 0. Recalling the ResNet structure with variable τ, (5.2), it is obvious that e.g. from $\tau^{[1]} \approx 0$ we can deduce $y^{[2]} \approx P_1^2 y^{[1]}$. Consequently, we delete the hidden layers 2,4 and 5, cf. Figure 2. The reduced network with 2 hidden layers achieves the same relative error on the test set, i.e. 0.07.
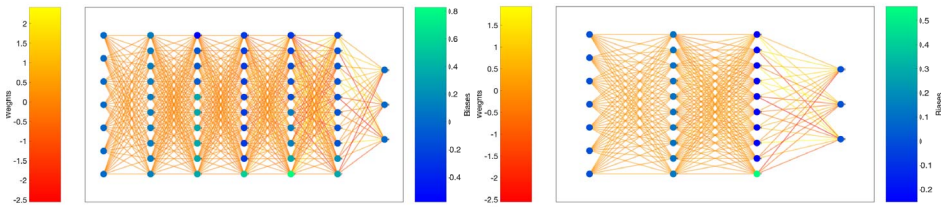
Figure 2: Left: Optimal weights and biases for ResNet with variable τ with 5 hidden layers and 10 nodes each with bias ordering. Right: Reduced ResNet with 2 hidden layers, i.e. hidden layers 1 and 3 from the larger network. The color of the dots indicates the bias value and the color of the lines indicates the magnitude of the weight.

While the same relative error is obtained with the reduced network, we aim at achieving even better results, therefore we next consider a larger network size with 6 hidden layers and 50 nodes in each layer (6-50). The proposed variable-τ approach seems to always outperform its constant τ counterpart, see Figure 1. As stated before, this is expected because the variable-τ framework and also Fractional-DNN have a bigger impact for deeper architectures with more hidden layers. Let us remark that the curves in Figure 1 are not monotone because we have only plotted the mean squared error term. In case of the entire J we do observe monotone behavior as expected. Even though we see in Table 1 that $\tau^{[\ell]} > 0$ for all ℓ in this setup, motivated by the reduction in the previous architecture of 5 hidden layers and 10 nodes per layer, instead of (6-50), we also consider a network with 2 hidden layers with 50 nodes per layer (2-50), which yields better results, cf. Figure 3.
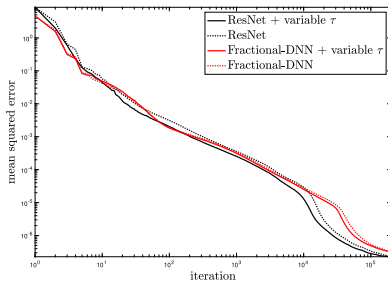
Results: 6 layers-50 nodes vs. 2 layers-50 nodes

From now on, $\boldsymbol{u}^{NN}(\boldsymbol{x})$ will denote the approximation of $\boldsymbol{u}$ obtained with a neural network at a point $\boldsymbol{x}$, the specific architecture will be clear from the context. Note that, $\boldsymbol{u}^{NN}(\boldsymbol{x}) = (\boldsymbol{u}_1^{NN}(\boldsymbol{x}), \boldsymbol{u}_2^{NN}(\boldsymbol{x}), \boldsymbol{u}_3^{NN}(\boldsymbol{x})) \in \mathbb{R}^3$.
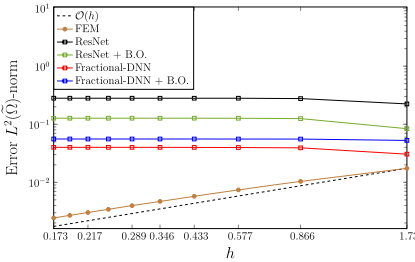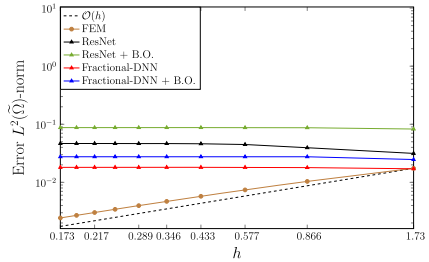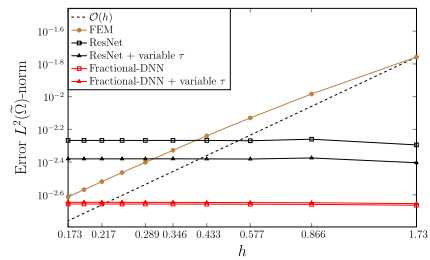

(a) Fixed τ − 6 layers, 50 nodes   (b) Variable τ − 6 layers, 50 nodes

(c) MSE plot − 2 layers, 50 nodes   (d) Comparison − 2 layers, 50 nodes

Figure 3: Comparison between various DNN architectures and FEM. **Top row:** L^2-error between an exact solution and DNN approximation (6 hidden layers with a width of 50 each) or FEM approximation. B.O. indicates bias ordering. The left and right panels correspond to fixed and variable τ, respectively. **Bottom row:** The left panel shows the mean squared error during training of different DNNs with 2 hidden layers with a width of 50 nodes each. The right panel displays the L^2-error between an exact solution and DNN approximation for the same DNNs and FEM.

We compare the neural network results with an approximation obtained with the FEM, see Figure 3(a), 3(b), and 3(d). To do that, we consider the unit cube $(0,1)^3$, denoted by $\widetilde{\Omega}$ as a domain. The unit cube is considered, to test how well the Neural Network performs for unseen data, and to test its extrapolation properties. Recall that the training data has been generated

Table 1: Optimal learned τ variables for various DNN architectures with τ-variable framework with 6 layers and 2 layers. These are the same network architectures that are considered in Figure 3

6-50	$\tau^{[0]}$	$\tau^{[1]}$	$\tau^{[2]}$	$\tau^{[3]}$	$\tau^{[4]}$	$\tau^{[5]}$	2-50	$\tau^{[0]}$	$\tau^{[1]}$
ResNet	0.92	0.95	0.99	0.95	0.92	0.88		0.84	0.87
ResNet, B.O.	0.70	0.94	1.00	0.96	0.86	0.70		0.28	0.51
Frac-DNN	0.59	0.70	0.53	0.34	0.28	0.30		0.94	0.93
Frac-DNN, B.O.	0.67	0.80	0.78	0.68	0.44	0.04		0.85	0.95

on Ω, which is cylindrical. For the FEM, we consider 10 uniform refinements of the unit cube $(0,1)^3$ and denote by h the mesh size of each one. Then, we compute $\|\boldsymbol{u} - \boldsymbol{u}^{NN}\|_{\widetilde{\Omega}}$ and $\|\boldsymbol{u} - \boldsymbol{u}_h\|_{\widetilde{\Omega}}$, where $\|\cdot\|_{\widetilde{\Omega}}$ denotes the $L^2(\widetilde{\Omega})^3$-norm and $\boldsymbol{u}_h$ denotes the best approximation of $\boldsymbol{u}$ into $\mathcal{N}_0(\widetilde{\Omega})$, with respect to the $H(\mathrm{curl};\widetilde{\Omega})$-norm.

In Figure 3(d), we observe that, for the DNN with 2 layers and for large h, the DNN approach gives a better approximation than the Lowest Order Nédélec space. We further notice that, as h gets smaller, $\boldsymbol{u}^{NN}$ needs to be evaluated at more points in $\widetilde{\Omega} \setminus \Omega$ and it is not obvious if the DNN approximation will remain stable. However, Figure 3(a), 3(b), and 3(d) show that the DNN approximation remains stable.

There exist several ways to measure how well a neural network performs. For instance, for the case of 2 layers and 50 nodes, the training error is slightly better with the ResNet-based architectures, cf. Figure 3(c), while a smaller error on unseen data is achieved with Fractional-DNNs, cf. Figure 3(d). Furthermore, when we plot the error on the square $(-1,1)^2 \times \{0.5\}$, we see that Fractional-DNNs, cf. Figure 4(c) and 4(d), extrapolate better than ResNets, cf. Figure 4(a) and 4(b). In the ResNet setting, employing the τ-learning framework (Figure 4(b)) yields to slightly better results than fixing τ (Figure 4(a)). Additionally, from its definition, we know $\boldsymbol{u}_3 \equiv 0$, cf. (7.2). Hence, we present quiver plots of the first two components of $\boldsymbol{u}(\cdot, x_3)$ for any x_3, and $\boldsymbol{u}^{NN}(\cdot, 0.5)$, in Figure 5. The two plots seem to coincide. Therefore, we further present pointwise errors restricted to Ω, where the pointwise error is measured in the $(\mathbb{R}^2)$ Euclidean norm. For Fractional-DNN with variable τ (2-50) with no bias ordering, inside Ω, we observe that $-0.0051 \le \boldsymbol{u}_3^{NN}(\boldsymbol{x}) \le 0.0077$, i.e., the constraints violation is of the order of approximation error. Meanwhile, ResNet with variable τ achieves better results in this test case.

Besides improving the training, the approximation could be further improved if we know a priori some qualitative properties of the exact solution. Then, they could be forced in the loss functional, similarly, as it is
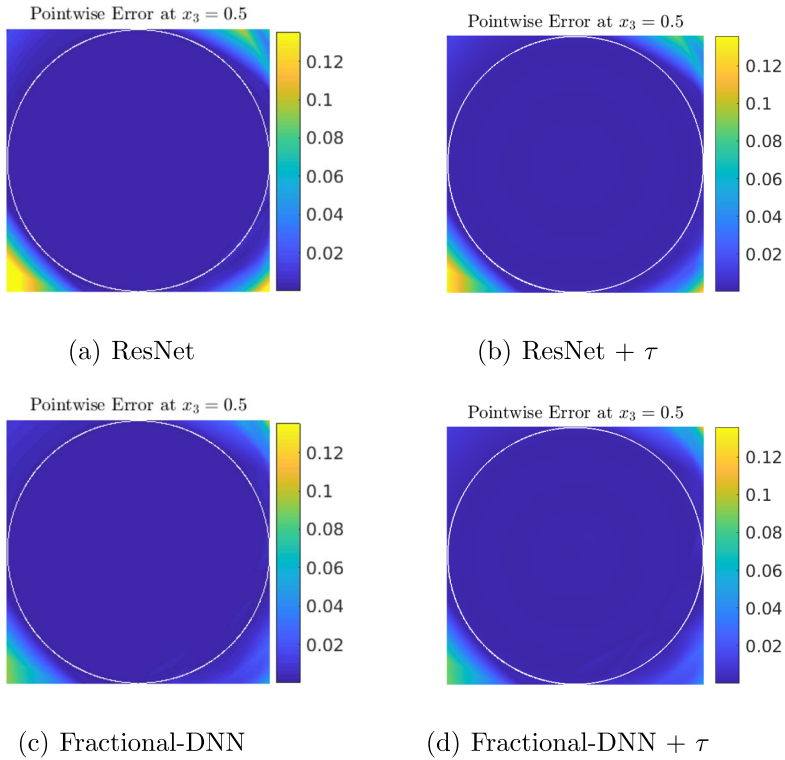
Figure 4: Comparison of testing errors between ResNet, ResNet with τ-learning framework, Fractional-DNN and Fractional-DNN with τ-learning framework (2-50).

done with PINNs, cf. [12]. It is important to mention that several other numerical examples were considered to test the robustness of our τ-variable framework. For instance, we considered problems where the standard Neural ODEs (cf. [16]) struggle to obtain good approximations, as pointed out in [21]. We obtained similar results to the ones presented here. For the sake of brevity, those results have been excluded.

Conclusion

A time variable learning framework for DNNs has been introduced, which in general can be applied to any DNN. However, from a mathematical perspective, DNN architectures which can be related to dynamical systems are of interest, since learning τ then corresponds to optimal adaptive time
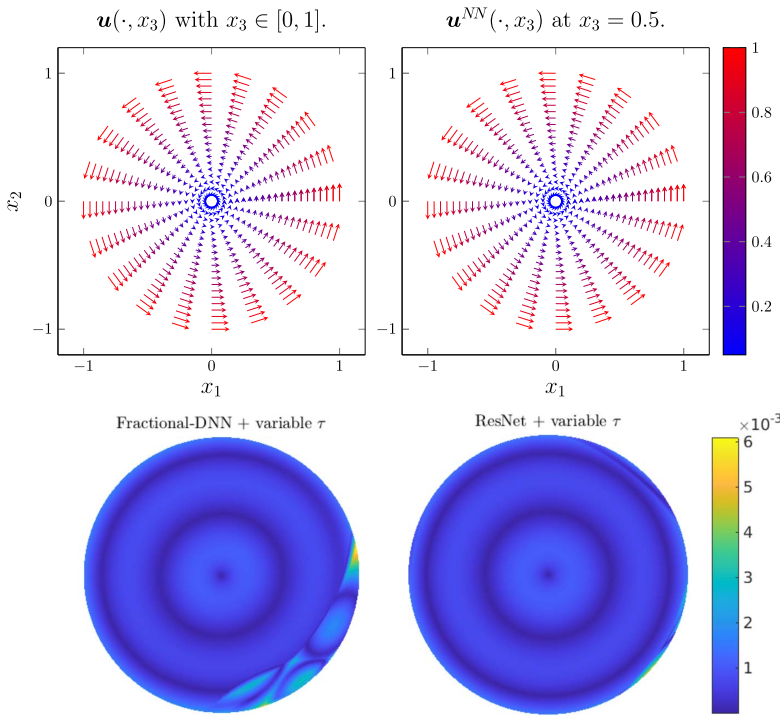
Figure 5: $\boldsymbol{u}$, $\boldsymbol{u}^{NN}$ and pointwise error on Ω, at $x_3 = 0.5$ ($x_1 x_2$-plane).

stepping. Consequently, special emphasis has been put on applying the τ-variable framework to ResNet and Fractional-DNN. The τ-variable framework is argued to overcome vanishing and exploding gradient challenges. The numerical results suggest that DNNs with τ-variable framework outperform their counterparts with fixed τ for deep architectures and enjoy an improved training error decay. Moreover, this method has the potential of identifying redundant layers, so that the network size can be reduced while maintaining the quality of the prediction.

Appendix A. Derivatives of the Lagrangian $\mathcal{L}$

We provide detailed calculations of derivatives needed in Section 5.2. To this end we recall

$$\mathcal{L}(y, \theta, \phi) = J(\theta) - \sum_{\ell=1}^{L-1} \langle y^{[\ell]} - P_{\ell-1}^{\ell} y^{[\ell-1]} + \sum_{j=0}^{\ell-2} a_{\ell-1,j} (P_{j+1}^{\ell} y^{[j+1]} - P_j^{\ell} y^{[j]})$$

$$- (\tau^{[\ell-1]})^\gamma c_{\gamma-1}^{-1} \sigma(W^{[\ell-1]}y^{[\ell-1]} + b^{[\ell-1]}), \phi^{[\ell]}\rangle$$

$$- \left\langle y^{[L]} - W^{[L-1]}y^{[L-1]}, \phi^{[L]} \right\rangle$$

A.1. Derivative with respect to $y^{[\ell]}$

Here, we calculate the variation of $\mathcal{L}$ with respect to $y^{[\ell]}$ for $\ell = 1, \ldots, L$:

$$\partial_{y^{[\ell]}} \mathcal{L}(y, \theta, \phi) = \partial_{y^{[\ell]}} J(\theta) - \partial_{y^{[\ell]}} \left(\sum_{k=1}^{L-1} \langle y^{[k]} - P_{k-1}^k y^{[k-1]}, \phi^{[k]} \rangle \right)$$

$$- \partial_{y^{[\ell]}} \left(\sum_{k=1}^{L-1} \left\langle \sum_{j=0}^{k-2} a_{k-1,j}(P_{j+1}^k y^{[j+1]} - P_j^k y^{[j]}), \phi^{[k]} \right\rangle \right)$$

$$+ \partial_{y^{[\ell]}} \left(\sum_{k=1}^{L-1} \left\langle (\tau^{[k-1]})^\gamma c_{\gamma-1}^{-1} \sigma(W^{[k-1]}y^{[k-1]} + b^{[k-1]}), \phi^{[k]} \right\rangle \right)$$

$$- \partial_{y^{[\ell]}} \left\langle y^{[L]} - W^{[L-1]}y^{[L-1]}, \phi^{[L]} \right\rangle.$$

From e.g. [5, Section 4.2.] we have most components of this expression already given. The main difference lies in the factors $a_{k-1,j}$, since the contained $\tau^{[j]}, \ldots, \tau^{[k-1]}$ are variable now. We only calculate the remaining unknown term, i.e. the second line in the above equation. First we rewrite the double sum in the following way:

$$- \sum_{k=1}^{L-1} \left\langle \sum_{j=0}^{k-2} a_{k-1,j}(P_{j+1}^k y^{[j+1]} - P_j^k y^{[j]}), \phi^{[k]} \right\rangle$$

$$= \sum_{k=0}^{L-2} \left(\sum_{j=0}^{k-1} a_{k,j} \langle P_j^{k+1} y^{[j]}, \phi^{[k+1]} \rangle - \sum_{j=0}^{k-1} a_{k,j} \langle P_{j+1}^{k+1} y^{[j+1]}, \phi^{[k+1]} \rangle \right)$$

$$= \sum_{k=0}^{L-2} \left(\sum_{j=0}^{k-1} a_{k,j} \langle P_j^{k+1} y^{[j]}, \phi^{[k+1]} \rangle - \sum_{j=1}^{k} a_{k,j-1} \langle P_j^{k+1} y^{[j]}, \phi^{[k+1]} \rangle \right).$$

Now we take the derivative with respect to $y^{[\ell]}$ and can apply the sum rule of differentiation:

$$\sum_{k=0}^{L-2} \sum_{j=0}^{k-1} a_{k,j} \, \partial_{y^{[\ell]}} \langle y^{[j]}, (P_j^{k+1})^\top \phi^{[k+1]} \rangle - \sum_{k=0}^{L-2} \sum_{j=1}^{k} a_{k,j-1} \, \partial_{y^{[\ell]}} \langle y^{[j]}, (P_j^{k+1})^\top \phi^{[k+1]} \rangle$$

$$= \sum_{k=0,\ell\leq k-1}^{L-2} a_{k,\ell}(P_\ell^{k+1})^\top \phi^{[k+1]} - \sum_{k=0,\ell\leq k}^{L-2} a_{k,\ell-1}(P_\ell^{k+1})^\top \phi^{[k+1]}$$

$$= \sum_{k=\ell+1}^{L-2} a_{k,\ell}(P_\ell^{k+1})^\top \phi^{[k+1]} - \sum_{k=\ell}^{L-2} a_{k,\ell-1}(P_\ell^{k+1})^\top \phi^{[k+1]}$$

For $\ell = L$ and $\ell = L - 1$ this derivative vanishes. For $\ell = 1, \ldots, L - 2$ we can slightly rewrite to get

$$\sum_{k=\ell+1}^{L-2} a_{k,\ell}(P_\ell^{k+1})^\top \phi^{[k+1]} - \sum_{k=\ell}^{L-2} a_{k,\ell-1}(P_\ell^{k+1})^\top \phi^{[k+1]}$$

$$= \sum_{k=\ell+2}^{L-1} (a_{k-1,\ell} - a_{k-1,\ell-1})(P_\ell^k)^\top \phi^{[k]} - a_{\ell,\ell-1}(P_\ell^{\ell+1})^\top \phi^{[\ell+1]}.$$

Now, the derivative can be assembled.

A.2. Derivative with respect to $\tau^{[\ell]}$

Care must be observed as $a_{k,j}$ contains $\tau^{[j]}, \ldots, \tau^{[k]}$. The derivative of $\mathcal{L}(y, \theta, \phi)$ with respect to $\tau^{[\ell]}$ for $\ell = 0, \ldots, L - 2$ therefore consists of the following terms:

$$\partial_{\tau^{[\ell]}} \mathcal{L}(y, \theta, \phi) = \partial_{\tau^{[\ell]}} J(\theta) - \partial_{\tau^{[\ell]}} \left(\sum_{k=1}^{L-1} \left\langle \sum_{j=0}^{k-2} a_{k-1,j}(P_{j+1}^k y^{[j+1]} - P_j^k y^{[j]}), \phi^{[k]} \right\rangle \right)$$

$$+ \partial_{\tau^{[\ell]}} \left(\sum_{k=1}^{L-1} \left\langle (\tau^{[k-1]})^\gamma c_{\gamma-1}^{-1} \sigma(W^{[k-1]} y^{[k-1]} + b^{[k-1]}), \phi^{[k]} \right\rangle \right).$$

We make an index shift from $k - 1$ to k, use the sum rule of differentiation and the fact that only $\tau^{[j]}, \ldots, \tau^{[k]}$ are contained in $a_{k,j}$ to obtain

$$\partial_{\tau^{[\ell]}} \left(-\sum_{k=1}^{L-1} \left\langle \sum_{j=0}^{k-2} a_{k-1,j}(P_{j+1}^k y^{[j+1]} - P_j^k y^{[j]}), \phi^{[k]} \right\rangle \right)$$

$$= -\sum_{k=0}^{L-2} \sum_{j=0}^{k-1} \partial_{\tau^{[\ell]}}(a_{k,j}) \left\langle P_{j+1}^{k+1} y^{[j+1]} - P_j^{k+1} y^{[j]}, \phi^{[k+1]} \right\rangle$$

$$= -\sum_{k=\ell}^{L-2} \sum_{j=0}^{\min\{k-1,\ell\}} \partial_{\tau^{[\ell]}}(a_{k,j}) \left\langle P_{j+1}^{k+1} y^{[j+1]} - P_j^{k+1} y^{[j]}, \phi^{[k+1]} \right\rangle.$$

Using the above equality, we arrive at

$$\partial_{\tau^{[\ell]}} \mathcal{L}(y, \theta, \phi)$$
$$= \partial_{\tau^{[\ell]}} J(\theta) - \sum_{k=\ell}^{L-2} \sum_{j=0}^{\min\{k-1,\ell\}} \partial_{\tau^{[\ell]}}(a_{k,j}) \left\langle P_{j+1}^{k+1} y^{[j+1]} - P_j^{k+1} y^{[j]}, \phi^{[k+1]} \right\rangle$$
$$+ \left\langle \gamma(\tau^{[\ell]})^{\gamma-1} c_{\gamma-1}^{-1} \sigma(W^{[\ell]} y^{[\ell]} + b^{[\ell]}), \phi^{[\ell+1]} \right\rangle.$$

For implementations we may want to understand the double summation in more detail. In fact, it can be split up into 3 terms in the following way:

$$-\sum_{k=\ell}^{L-2} \sum_{j=0}^{\min\{k-1,\ell\}} \partial_{\tau^{[\ell]}}(a_{k,j}) \left\langle P_{j+1}^{k+1} y^{[j+1]} - P_j^{k+1} y^{[j]}, \phi^{[k+1]} \right\rangle$$
$$= -\sum_{j=0}^{\ell-1} \partial_{\tau^{[\ell]}}(a_{\ell,j}) \left\langle P_{j+1}^{\ell+1} y^{[j+1]} - P_j^{\ell+1} y^{[j]}, \phi^{[\ell+1]} \right\rangle$$
$$- \sum_{k=\ell+1}^{L-2} \sum_{j=0}^{\ell-1} \partial_{\tau^{[\ell]}}(a_{k,j}) \left\langle P_{j+1}^{k+1} y^{[j+1]} - P_j^{k+1} y^{[j]}, \phi^{[k+1]} \right\rangle$$
$$- \sum_{k=\ell+1}^{L-2} \partial_{\tau^{[\ell]}}(a_{k,\ell}) \left\langle P_{\ell+1}^{k+1} y^{[\ell+1]} - P_{\ell}^{k+1} y^{[\ell]}, \phi^{[k+1]} \right\rangle.$$

Now for each of the above terms we can easily compute the contained derivative using basic differentiation rules. Employing the product rule for $j = 0, \ldots, \ell - 1$, i.e. $j < \ell$, we get

$$\partial_{\tau^{[\ell]}}(a_{\ell,j}) = (1 - \gamma) \frac{(\tau^{[\ell]})^{\gamma}}{\tau^{[j]}} \left(\left(\sum_{i=j}^{\ell} \tau^{[i]} \right)^{-\gamma} - \left(\sum_{i=j+1}^{\ell} \tau^{[i]} \right)^{-\gamma} \right)$$
$$+ \gamma \frac{(\tau^{[\ell]})^{\gamma-1}}{\tau^{[j]}} \left(\left(\sum_{i=j}^{\ell} \tau^{[i]} \right)^{1-\gamma} - \left(\sum_{i=j+1}^{\ell} \tau^{[i]} \right)^{1-\gamma} \right).$$

For $j = 0, \ldots, \ell - 1$ and $k = \ell + 1, \ldots, L - 2$, i.e. $j < \ell < k$, we have

$$\partial_{\tau^{[\ell]}}(a_{k,j}) = (1 - \gamma)\frac{(\tau^{[k]})^\gamma}{\tau^{[j]}}\left(\left(\sum_{i=j}^{k} \tau^{[i]}\right)^{-\gamma} - \left(\sum_{i=j+1}^{k} \tau^{[i]}\right)^{-\gamma}\right).$$

And finally, using the product rule again for $k = \ell + 1, \ldots, L - 2$, i.e. $\ell < k$, we see

$$\partial_{\tau^{[\ell]}}(a_{k,\ell}) = \frac{(\tau^{[k]})^\gamma}{(\tau^{[\ell]})^2}\left(\left(\sum_{i=\ell+1}^{k} \tau^{[i]}\right)^{1-\gamma} - \left(\sum_{i=\ell}^{k} \tau^{[i]}\right)^{1-\gamma}\right)$$
$$+ (1 - \gamma)\frac{(\tau^{[k]})^\gamma}{\tau^{[\ell]}}\left(\sum_{i=\ell}^{k} \tau^{[i]}\right)^{-\gamma}$$

References

[1] Harbir Antil, Livia Betz, and Daniel Wachsmuth. Strong stationarity for optimal control problems with non-smooth integral equation constraints: Application to continuous dnns. *To appear: Applied Mathematics and Optimization*, 2023.

[2] Harbir Antil, Thomas S. Brown, Rainald Löhner, Fumiya Togashi, and Deepanshu Verma. Deep neural nets with fixed bias configuration. *Numerical Algebra, Control and Optimization*, 2022.

[3] Harbir Antil, Howard C Elman, Akwum Onwunta, and Deepanshu Verma. A deep neural network approach for parameterized pdes and bayesian inverse problems. *Machine Learning: Science and Technology*, 4(3):035015, Aug 2023.

[4] Harbir Antil, Ciprian G. Gal, and Mahamadi Warma. A unified framework for optimal control of fractional in time subdiffusive semilinear pdes. *Discrete and Continuous Dynamical Systems – Series S*, 10 2021. MR4438772

[5] Harbir Antil, Ratna Khatri, Rainald Löhner, and Deepanshu Verma. Fractional deep neural network via constrained optimization. *Machine Learning: Science and Technology*, 2(1):015003, 2020.

[6] Harbir Antil, Drew P. Kouri, Martin-D. Lacasse, and Denis Ridzal, editors. *Frontiers in PDE-constrained optimization*, volume 163 of *The*

...

IMA Volumes in Mathematics and its Applications. Springer, New York, 2018. Papers based on the workshop held at the Institute for Mathematics and its Applications, Minneapolis, MN, June 6–10, 2016. MR3839310

[7] Harbir Antil and Deepanshu Verma. fractional-DNN, May 2023. https://github.com/Center-for-Math-AI/Fractional_DNNs.git.

[8] Trevor Avant and Kristi A. Morgansen. Analytical bounds on the local lipschitz constants of relu networks. *ArXiv*, 2104.14672, 2021. MR4605342

[9] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[10] Martin Benning, Elena Celledoni, Matthias J. Ehrhardt, Brynjulf Owren, and Carola-Bibiane Schönlieb. Deep learning as optimal control problems: Models and numerical methods. *Journal of Computational Dynamics*, 6(2):171–198, 2019. MR4043228

[11] Benjamin Bischke, Prakriti Bhardwaj, Aman Gautam, Patrick Helber, Damian Borth, and Andreas Dengel. Detection of flooding events in social multimedia and satellite imagery using deep neural networks. In *MediaEval*, 2017.

[12] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (pinns) for fluid mechanics: A review, 2021. MR3995303

[13] Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[14] Hao Chen, Qi Dou, Lequan Yu, Jing Qin, and Pheng-Ann Heng. Voxresnet: Deep voxelwise residual networks for brain segmentation from 3D MR images. *NeuroImage*, 170:446–455, 2018. Segmenting the Brain.

[15] Kunjin Chen, Kunlong Chen, Qin Wang, Ziyu He, Jun Hu, and Jinliang He. Short-term load forecasting with deep residual networks. *IEEE Transactions on Smart Grid*, 10(4):3943–3952, 2018.

[16] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019.

[17] Patrick Ciarlet, Haijun Wu, and Jun Zou. Edge element methods for Maxwell's equations with strong convergence for Gauss' laws. *SIAM J. Numer. Anal.*, 52(2):779–807, 2014. MR3188392

[18] Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. Adanet: Adaptive structural learning of artificial neural networks. In *International Conference on Machine Learning*, pages 874–883. PMLR, 2017. MR3590993

[19] Ronald DeVore, Boris Hanin, and Guergana Petrova. Neural network approximation. *Acta Numer.*, 30:327–444, 2021. MR4298220

[20] Kai Diethelm and Neville J. Ford. Analysis of fractional differential equations. *J. Math. Anal. Appl.*, 265(2):229–248, 2002. MR1876137

[21] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes, 2019.

[22] Weinan E. A proposal on machine learning via dynamical systems. *Commun. Math. Stat.*, 5(1):1–11, 2017. MR3627592

[23] Weinan E. Machine learning: Mathematical theory and scientific applications. *Notices of the American Mathematical Society*, 66(11):1813–1820, 2019. MR3971087

[24] Robert M. Farber, Alan S. Lapedes, Ramiro Rico-Martínez, and Ioannis G. Kevrekidis. Identification of continuous-time dynamical systems: Neural network based algorithms and parallel implementation. In Richard F. Sincovec, David E. Keyes, Michael R. Leuze, Linda R. Petzold, and Daniel A. Reed, editors, *PPSC*, pages 287–291. SIAM, 1993.

[25] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. MR3617773

[27] Jayadeep Gopalakrishnan, Luis E. García-Castillo, and Leszek F. Demkowicz. Nédélec spaces in affine coordinates. *Comput. Math. Appl.*, 49(7-8):1285–1294, 2005. MR2141266

[28] Stefanie Gunther, Lars Ruthotto, Jacob B Schroder, Eric C Cyr, and Nicolas R Gauger. Layer-parallel training of deep residual neural net-

works. *SIAM Journal on Mathematics of Data Science*, 2(1):1–23, 2020. MR4060448

[29] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017. MR3742361

[30] Kerstin Hammernik, Teresa Klatzer, Erich Kobler, Michael P Recht, Daniel K Sodickson, Thomas Pock, and Florian Knoll. Learning a variational network for reconstruction of accelerated MRI data. *Magnetic Resonance in Medicine*, 79(6):3055–3071, 2018. MR3706322

[31] Soufiane Hayou, Eugenio Clerico, Bobby He, George Deligiannidis, Arnaud Doucet, and Judith Rousseau. Stable resnet. In *International Conference on Artificial Intelligence and Statistics*, pages 1324–1332. PMLR, 2021.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 630–645. Springer, 2016.

[35] Michael Hinze, René Pinnau, Michael Ulbrich, and Stefan Ulbrich. *Optimization with PDE Constraints*, volume 23 of *Mathematical Modelling: Theory and Applications*. Springer, New York, 2009. MR2516528

[36] Qingguo Hong, Jonathan W. Siegel, and Jinchao Xu. A priori analysis of stable neural network solutions to numerical PDEs, 2021.

[37] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

[38] Ameya D. Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and

physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020. MR4051868

[39] Kyong Hwan Jin, Michael T McCann, Emmanuel Froustey, and Michael Unser. Deep convolutional neural network for inverse problems in imaging. *IEEE Transactions on Image Processing*, 26(9):4509–4522, 2017. MR3670561

[40] Eurika Kaiser, J. Nathan Kutz, and Steven L. Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proc. A.*, 474(2219):20180335, 25, 2018. MR3895998

[41] Anatoly A. Kilbas, Hari M. Srivastava, and Juan J. Trujillo. *Theory and applications of fractional differential equations*, volume 204 of *North-Holland Mathematics Studies*. Elsevier Science B.V., Amsterdam, 2006. MR2218073

[42] Dongwook Lee, Jaejun Yoo, Sungho Tak, and Jong Chul Ye. Deep residual learning for accelerated mri using magnitude and phase networks. *IEEE Transactions on Biomedical Engineering*, 65(9):1985–1995, 2018.

[43] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021. MR4582511

[44] Yumin Lin, Xianjuan Li, and Chuanju Xu. Finite difference/spectral approximations for the fractional cable equation. *Math. Comp.*, 80(275):1369–1396, 2011. MR2785462

[45] Yumin Lin and Chuanju Xu. Finite difference/spectral approximations for the time-fractional diffusion equation. *J. Comput. Phys.*, 225(2):1533–1552, 2007. MR2349193

[46] Hailiang Liu and Peter Markowich. Selection dynamics for deep neural networks. *Journal of Differential Equations*, 269(12):11540–11574, 2020. MR4152217

[47] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pages 3276–3285. PMLR, 2018.

[48] Jean-Claude Nédélec. Mixed finite elements in $\mathbb{R}^3$. *Numer. Math.*, 35(3):315–341, 1980. MR0592160

[49] Ricardo H. Nochetto, Enrique Otárola, and Abner J. Salgado. A PDE approach to space-time fractional parabolic problems. *SIAM J. Numer. Anal.*, 54(2):848–873, 2016. MR3478958

[50] Fernando J. Pineda. Generalization of back-propagation to recurrent neural networks. *Phys. Rev. Lett.*, 59:2229–2232, 1987. MR0913691

[51] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29(9):2352–2449, 2017. MR3866781

[52] Wolfgang Ring and Benedikt Wirth. Optimization methods on Riemannian manifolds and their application to shape space. *SIAM J. Optim.*, 22(2):596–627, 2012. MR2968868

[53] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[54] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, 62(3):352–364, 2020. MR4082375

[55] Carola-Bibiane Schoenlieb, Martin Benning, Matthias Ehrhardt, Brynjulf Owren, and Elena Celledoni. Research data supporting" deep learning as optimal control problems". 2019. MR4043228

[56] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[57] Sho Sonoda and Noboru Murata. Transport analysis of infinitely deep neural network. *Journal of Machine Learning Research*, 20(2):1–52, 2019. MR3911409

[58] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. *arXiv preprint arXiv:1507.06228*, 2015.

[59] Ying Tai, Jian Yang, and Xiaoming Liu. Image super-resolution via deep recursive residual network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3147–3155, 2017.

[60] Vasily E. Tarasov. Differential equations with fractional derivative and universal map with memory. *J. Phys. A*, 42(46):465102, 13, 2009. MR2552008

[61] Quyen Tran, Harbir Antil, and Hugo Díaz. Optimal control of parameterized stationary maxwell's system: Reduced basis, convergence analysis, and a posteriori error estimates. *Mathematical Control and Related Fields*, 2022. MR4518560

[62] Songtao Wu, Shenghua Zhong, and Yan Liu. Deep residual learning for image steganalysis. *Multimedia Tools and Applications*, 77(9):10437–10453, 2018.

[63] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4541–4550, 2019.

[64] Qiang Zhang, Qiangqiang Yuan, Chao Zeng, Xinghua Li, and Yancong Wei. Missing data reconstruction in remote sensing image with a unified spatial–temporal–spectral deep convolutional neural network. *IEEE Transactions on Geoscience and Remote Sensing*, 56(8):4274–4288, 2018.

HARBIR ANTIL
THE CENTER FOR MATHEMATICS AND ARTIFICIAL INTELLIGENCE (CMAI)
AND DEPARTMENT OF MATHEMATICAL SCIENCES
GEORGE MASON UNIVERSITY
FAIRFAX, VA 22030
USA
E-mail address: hantil@gmu.edu

HUGO DÍAZ
DEPARTMENT OF MATHEMATICAL SCIENCES
UNIVERSITY OF DELAWARE
NEWARK, DE 19716
USA
E-mail address: hugodiaz@udel.edu

EVELYN HERBERG
THE CENTER FOR MATHEMATICS AND ARTIFICIAL INTELLIGENCE (CMAI)
AND DEPARTMENT OF MATHEMATICAL SCIENCES
GEORGE MASON UNIVERSITY
FAIRFAX, VA 22030
USA
E-mail address: evelyn.herberg@iwr.uni-heidelberg.de