भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

B. Tech Project

Searching for possibility of Neural Fractional Differential Equations

Supervisor : **Dr. P.K. Srijith**
Co-Supervisor : **Dr. Nithyanandan Kanagaraj**

Amaan
EP20BTECH11003

Acknowledgement

I would like to express my sincere gratitude towards my supervisor, **Prof. Srijith Sir** for giving me an opportunity to work under him. I got to learn a lot and explore the lesser known fields of Neural ODEs, Fractional Calculus and Fractional DNNs.
I would also like to thank **Prof. Nithyanandan Sir** for being a co-supervisor for my project and helping me through the project with deadlines and management.
I was fortunate to have them as my supervisors, who let me work at my own pace, had patience while I was in bad situations, guided me through the project with his reviews and suggestions, and most importantly, showed faith in me.

Amaan

ABSTRACT

After the advent of Neural Ordinary differential equations and Adjoint sensitivity method, we begin our search for Neural Fractional differential equations. We begin with gaining insights on fractional calculus and fractional differential equations. Moreover, we gain insights into novel Fractional Deep Neural Networks, which is potentially the discrete setting of Fractional Differential Equations which we intended to find.

Project Work

1 Fractional Deep Neural Networks

Fractional DNNs allows us to incorporate history(or memory) into the network in a mathematically rigorous manner, moreover, layers are fully connected. Fractional-DNN can be viewed as a time-discretization of a fractional in time nonlinear ordinary differential equation (ODE). Derivation of backpropagation and design equations is done using Lagrangian approach. The key benefits are a significant improvement to the vanishing gradient issue due to the memory effect, and better handling of nonsmooth data due to the network's ability to approximate non-smooth functions.

1.1 Preliminaries

Softmax function is defined as,

$$S(W,Y) = \exp(WY)\text{diag}\left(\frac{1}{e^T\exp(\text{WY})}\right) \tag{1}$$

and the cross entropy loss function is,

$$E(W,Y,C_{obs}) = -\frac{1}{n}\text{tr }(C_{obs}^T log(S(W,Y))) \tag{2}$$

Left hand Caputo Fractional Derivative is defined as,

$$d_t^\gamma = \frac{1}{\Gamma(1-\gamma)}\frac{d}{dt}\int_0^t \frac{u(r)-u(0)}{(t-r)^\gamma}dr \tag{3}$$

Right hand Caputo Fractional Derivative is defined as,

$$d_{T-t}^\gamma = -\frac{1}{\Gamma(1-\gamma)}\frac{d}{dt}\int_t^T \frac{u(r)-u(T)}{(r-t)^\gamma}dr \tag{4}$$

For a fixed real number $0 < \gamma < 1$, and an absolutely continuous function $u : [0,T] \to \mathbb{R}$

TABLE 1. Table of Notations.

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

1.2 Continuous Fractional DNNs

1.2.1 Classical RNN

A classical RNN helps approximate F , for a known set of inputs and outputs. To construct an RNN, for each layer j, we first consider a linear-transformation of Y_{j1} as,

$$G_{j-1}(Y_{j-1}) = K_{j-1}Y_{j-1} + b_{j-1} \tag{5}$$

Next we introduce non linearity using a nonlinear activation function σ (e.g. ReLU or tanh). The resulting RNN is,

$$Y_j = Y_{j-1} + \tau(\sigma \circ G_{j-1})(Y_{j-1}), \quad j = 1 \dots N; \quad N > 1 \tag{6}$$

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

$$F_\theta(.) = (I + \tau(\sigma \circ G_{N-1})(Y_{N-1})) \circ (I + \tau(\sigma \circ G_{N-2})(Y_{N-2})) \circ \cdots \circ (I + \tau(\sigma \circ G_0)(Y_0)) \tag{7}$$

where $\theta = (K_j, b_j)$. We need to learn θ and minimize the loss function $J(\theta, (Y_N, C))$ subject to constraint (6).

$$\min_{\theta} J(\theta, (Y_N, C)) \tag{8}$$

This system is a forward-Euler discretization of continuous in time ODE,

$$d_t Y(t) = \sigma(K(t)Y(t) + b(t)), \quad Y(0) = Y_0 \tag{9}$$

The continuous learning problem then requires minimizing the loss function J at the final time T subject to the ODE constraints (9):

$$\min_{\theta=(K,b)} J(\theta, (Y(T), C)) \tag{10}$$

Continuous DNNs will lead to algorithms which are independent of the neural network architecture, i.e., independent of the number of layers. Stability can be gauged and for NN (7), it has been noted that as the information about the input or gradient passes through many layers, it can vanish and "wash out", or grow and "explode" exponentially.

Notice that (8), and its discrete version (6), incorporates many algorithmic processes such as linear solvers, preconditioners, nonlinear solvers, optimization solvers, etc. Furthermore, there are well-established numerical algorithms that re-use information from previous iterations to accelerate convergence.

These methods account for the history $Y_j, Y_{j-1}, \ldots, Y_0$ while choosing Y_{j+1}. Motivated by these observations we introduce versions of (6) and (8) that can account for history (or memory) effects in a rigorous mathematical fashion.

1.2.2 Continuous Fractional-DNN

The fractional time derivative in (3) has a distinct ability to allow a memory effect. We use the idea of fractional time derivative to enrich the constraint optimization problem (9), and subsequently (9), by replacing the standard time derivative d_t by the fractional time derivative d_t^γ of order $\gamma \in (0, 1)$ Recall that for $\gamma = 1$, we obtain the classical derivative d_t. Our new continuous in time model, the Fractional-DNN, is then given by,

$$d_t^\gamma Y(t) = F_\theta(Y(t), t, \theta_t), Y(0) = Y_0 \tag{11}$$

1.2.3 Continuous Fractional-DNN and Cross Entropy Loss Functional

We consider classification problems. Using the cross entropy loss functional E, defined in (2), Replacing J in (10) by E together with a regularization term $R(W, K(t), b(t))$

$$\min_{W,K,b} E(W, Y(T), C_{obs}) + R(W, K(t), b(t)) \tag{12}$$

$$\text{s.t.} \quad d_t Y(t) = \sigma(K(t)Y(t) + b(t)), \quad Y(0) = Y_0 \tag{13}$$

To solve (12) and (13), we rewrite this problem as an unconstrained optimization problem via the Lagrangian functional and derive the optimality conditions. Let P denote the Lagrange multiplier, then the Lagrangian functional is given by,

$$L(Y, W, K, b; P) = E(W, Y(T), C_{obs}) + R(W, K(t), b(t)) + \langle d_t Y(t) - \sigma(K(t)Y(t) + b(t) \tag{14}$$

where $\langle ., . \rangle = \int_0^T \langle ., . \rangle_F dt$ is L^2-inner product and $\langle ., . \rangle_F$ the Frobenius Inner Product. Using fractional integration by parts, we get three set of necessary first order optimality conditions for a stationary state $(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P})$,

(A) **State Equation**

The gradient of L with respect to P at $(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P})$ yields the state equation, $\boldsymbol{\nabla}_P L(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P}) = 0$ or equivalently,

$$d_t \overline{Y}(t) = \sigma(\overline{K}(t)\overline{Y}(t) + \overline{b}(t)), \quad \overline{Y}(0) = \overline{Y}_0 \tag{15}$$

For the state variable $\overline{Y}$, we solve it forward in time, hence this equation is called Forward propagation.

(B) **Adjoint Equation**

The gradient of L with respect to Y at $(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P})$ yields the Adjoint Equation, $\boldsymbol{\nabla}_Y L(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P}) = 0$ or equivalently,

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

Adjoint equation is also called backward propagation.

(C) **Design Equations**

Equating $\boldsymbol{\nabla}_W L(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P}), \boldsymbol{\nabla}_K L(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P}), \boldsymbol{\nabla}_b L(\overline{Y}, \overline{W}, \overline{K}, \overline{b}; \overline{P})$ to zero, we get the design equations with $(\overline{W}, \overline{K}, \overline{b}; \overline{P})$ as the design variables,

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

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

1.3 Discrete Fractional Deep Neural Network

1.3.1 Approximation of Caputo Derivative

L^1-scheme is used to discretize the left and right Caputo fractional derivative. Consider the following fractional differential equation involving the left Caputo fractional derivative,

$$d_t^\gamma u(t) = f(u(t)), \quad u(0) = u_0 \tag{16}$$

Discretizing the time interval $[0, T]$ uniformly with step size τ ,

$$0 = t_0 < t_1 < t_2 < \cdots < t_{j+1} < \cdots < t_N = T, \quad t_j = j\tau \tag{17}$$

Then using L^1-scheme, discretization of (16) is given by,

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

where coefficients a_k are given by,

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

Next, the fractional differential equation involving the right Caputo fractional operator is discretized for $0 < \gamma < 1$,

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

Again using L^1-scheme, we get the following discretization of (20),

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

1.3.2 Discrete Optimality Conditions

Notice that, each time-step corresponds to one layer of the neural network. It is necessary to do one forward propagation (state solve) and one backward propagation (adjoint solve) to derive an expression of the gradient with respect to the design variables.

(A) **Discrete State Equation** We again use L^1-scheme to discretize,

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

(B) **Discrete Adjoint Equation** We again use L^1-scheme to discretize,

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

(C) **Discrete Gradient w.r.t. Design Variables** We again use L^1-scheme to discretize,

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

1.4 Fractional-DNN Algorithm

Fractional-DNN is a supervised learning architecture, i.e. it comprises of a training phase and a testing phase.

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

Input: $(Y_0, C_{obs})\,, W,\ \{K_j, b_j\}_{j=0}^{N-1},\ N,\ \tau,\ \gamma$
Output: $\{Y_j\}_{j=1}^{N},\ P_N,$

Let $z_0 = 0$.
for $j = 1, \cdots, N$ **do**
  for $k = 1, \cdots, j-1$ **do**
    Compute a_{j-k}: {Use (19)}
    Update z_k: $z_k = z_{k-1} + a_{j-k}\,(Y_k - Y_{k-1})$
  end for
  Update Y_j: $Y_j \ = \ Y_{j-1} - z_{j-1} + (\tau)^\gamma\,\Gamma(2-\gamma)\,\sigma(K_{j-1}Y_{j-1} + b_{j-1})$
end for
Compute P_N: $P_N = -(n)^{-1}\,W^\intercal(-C_{obs} + S(W, Y_N))$

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

Input: $\{Y_j\}_{j=1}^{N},\ P_N,\ \{K_j, b_j\}_{j=0}^{N-1},\ N,\ \tau,\ \gamma$
Output: $\{P_j\}_{j=0}^{N-1}$

Let $x_0 = 0$.
for $j = N-1, \cdots, 0$ **do**
  for $k = j+1, \cdots, N-1$ **do**
    Compute a_{k-j-1}: {Use (19)}
    Compute x_k: $x_k = x_{k-1} + a_{k-j-1}\,(P_{k+1} - P_k)$
  end for
  Update P_j: $P_j = P_{j+1} + x_{N-1} - (\tau)^\gamma\,\Gamma(2-\gamma)\,[-K_j^\intercal(P_{j+1} \odot \sigma'(K_jY_{j+1} + b_j))]$
end for

Algorithm 3 Training Phase of Factional-DNN

Input: (Y_0, C_{obs}) , N, τ, γ, m_1, m_2
Output: W, $\{K_j, b_j\}_{j=0}^{N-1}$, $C_{train}, \alpha_{train}$,

1: Initialize $W, \{K_j, b_j\}_{j=0}^{N-1}$
2: **for** $i = 1, \cdots, m_1$ **do**
3: Let $(\hat{Y}_0, \hat{C}_{obs}) \subset (Y_0, C_{obs})$ {Randomly select a mini-batch and apply BN using (27)}
4: **FORWARD PROPAGATION** {Use Algorithm 1 to get $\{\hat{Y}_j\}_{j=1}^{N}$, P_N}.
5: **BACKWARD PROPAGATION** {Use Algorithm 2 to get $\{P_j\}_{j=0}^{N-1}$}.
6: **GRADIENT COMPUTATION**
7: Compute $\nabla_W \mathcal{L}$, $\{\nabla_K \mathcal{L}\}$, $\{\nabla_b \mathcal{L}\}$

$$\begin{aligned}
\nabla_W \mathcal{L} &= (n)^{-1} \left(-C_{obs} + S(W, \hat{Y}_N)\right) \ (\hat{Y}_N)^\intercal + \nabla_W \mathcal{R}(W, K_j, b_j) \\
\nabla_K \mathcal{L} &= -\hat{Y}_j \ \left(P_{j+1} \odot \sigma'(K_j \hat{Y}_j + b_j)\right)^\intercal + \nabla_K \mathcal{R}(W, K_j, b_j) \\
\nabla_b \mathcal{L} &= -tr\left(\sigma'(K_j \hat{Y}_j + b_j) \ P_{j+1}\right) + \nabla_b \mathcal{R}(W, K_j, b_j)
\end{aligned}$$

8: Pass $\nabla_W \mathcal{L}$, $\nabla_K \mathcal{L}$, $\nabla_b \mathcal{L}$ to gradient based solver with m_2 max iterations to update $W, \{K_j, b_j\}_{j=0}^{N-1}$.
9: Compute $\hat{C}_{train} = S(W, \hat{Y}_N)$
10: Compare $\hat{C}_{train}$ to $\hat{C}_{obs}$ to compute α_{train}
11: **end for**

Algorithm 4 Testing Phase of Fractional-DNN

Input: $(Y_0^{test}, C_{obs,test})$, W, $\{K_j, b_j\}_{j=0}^{N-1}$, N, τ, γ
Output: C_{test}, α_{test}

1: Let $Y_0 = Y_0^{test}$ {Apply BN using (27)}
2: **FORWARD PROPAGATION** {Use Algorithm 1 to get $\{Y_j\}_{j=1}^{N}$}.
3: Compute $C_{test} = S(W, Y_N)$
4: Compare C_{test} to $C_{obs,test}$ to compute α_{test}

References

- Neural Ordinary Differential Equations
- Fractional Differential Equations by Igor Podlubny
- Fractional DNNs