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, through which I could learn and explore the field of Reference based Data Compression in the domain of Source Coding.
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 differntial equations. We begin with gaining insights on fractional calculus and fractional differential equations.

# Project Work

The project begins with a whole new family of Deep Neural Networks, Neural Ordinary Differential Equations, upon which we plan to build.

## 1 Neural Ordinary Differential Equations

Instead of specifying a discrete sequence of hidden layers, the derivative of the hidden state are parameterized using a neural network. The output of the network is computed using a black-box differential equation solver. These continuous depth models have constant memory cost, adapt their evaluation strategy to each input, and can explicitly trade numerical precision for speed.

### 1.1 Introduction

Models such as residual networks, recurrent neural network decoders, build complicated transformations by composing a sequence of transformations to a hidden state,

$$h_{t+1} = h_t + f(h_t, \theta) \tag{1}$$

Slight manipulation yields,

$$\frac{h_{t+1} - h_t}{1} = f(h_t, \theta) \tag{2}$$

$$\frac{h_{t+\Delta} - h_t}{\Delta} = f(h_t, \theta) \qquad : \Delta = 1 \tag{3}$$

Now, we start decreasing the step size $\Delta$, in the limit, we parameterize the continuous dynamics of hidden units using an ordinary differential equation (ODE) specified by a neural network,

$$\lim_{\Delta \to 0} \frac{h(t+\Delta) - h(t)}{\Delta} = \frac{dh(t)}{dt} = f(h(t), t, \theta) \tag{4}$$

h(0) and h(T) are defined as input and output layers connected by the curve traced by (4) which is computed by a black-box differential equation solver.

## 1.2 Reverse-mode automatic differentiation of ODE solutions

The main technical difficulty in training continuous-depth networks is performing reverse-mode differentiation(backpropagation) through the ODE solver. Differentiating through the operations of the forward pass is straightforward, but incurs a high memory cost and introduces additional numerical error.

In traditional neural networks, for backpropagation, we need to store the results obtained from forward propagation, which in this case is too costly, because the number of layers and hence the number of parameters is virtually infinite (few million). RAM will blow up!

Hence, we use *Adjoint Sensitivity Method* to compute Gradients. This approach computes gradients by solving a second, augmented ODE backwards in time, and is applicable to all ODE solvers. This approach scales linearly with problem size, has low memory cost, and explicitly controls numerical error.

Consider optimizing a scalar-valued loss function L(),

$$L(\mathbf{z}(t_1)) = L\Big(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\Big) = L(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)) \tag{5}$$

To optimize $L$, we require gradients w.r.t $\theta$, first we determine the gradient w.r.t $\mathbf{z}(\mathbf{t})$,the hidden state, which is defined as **adjoint**, $\mathbf{a}(t) = \partial L/\partial \mathbf{z}(t)$, Its dynamics are given by another ODE,

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^T \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}} \tag{6}$$

We can compute $\partial L/\partial \mathbf{z}(t_0)$ by another call to ODE solver. This solver must run backwards, starting from the initial value of $\partial L/\partial \mathbf{z}(t_1)$. Also,

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(t)^T \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}} dt \tag{7}$$

Here is the algorithm to construct the necessary dynamics, and call an ODE solver to compute all gradients at once.

**Algorithm 1** Reverse-mode derivative of an ODE initial value problem

**Input:** dynamics parameters $\theta$, start time $t_0$, stop time $t_1$, final state $\mathbf{z}(t_1)$, loss gradient $\partial L / \partial \mathbf{z}(t_1)$

$s_0 = [\mathbf{z}(t_1), \frac{\partial L}{\partial \mathbf{z}(t_1)}, \mathbf{0}_{|\theta|}]$ ▷ Define initial augmented state

**def** aug_dynamics($[\mathbf{z}(t), \mathbf{a}(t), \cdot], t, \theta$): ▷ Define dynamics on augmented state

  **return** $[f(\mathbf{z}(t), t, \theta), -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial \mathbf{z}}, -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial \theta}]$ ▷ Compute vector-Jacobian products

$[\mathbf{z}(t_0), \frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}] = \text{ODESolve}(s_0, \text{aug\_dynamics}, t_1, t_0, \theta)$ ▷ Solve reverse-time ODE

**return** $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}$ ▷ Return gradients

## 1.3 Experiments on Neural ODEs and Performance

### 1.3.1 Performance on MNIST dataset

Table 1: Performance on MNIST. †From LeCun et al. (1998).

| | Test Error | # Params | Memory | Time |
|---|---|---|---|---|
| 1-Layer MLP† | 1.60% | 0.24 M | - | - |
| ResNet | 0.41% | 0.60 M | $\mathcal{O}(L)$ | $\mathcal{O}(L)$ |
| RK-Net | 0.47% | 0.22 M | $\mathcal{O}(\tilde{L})$ | $\mathcal{O}(\tilde{L})$ |
| ODE-Net | 0.42% | 0.22 M | $\boldsymbol{\mathcal{O}(1)}$ | $\mathcal{O}(\tilde{L})$ |

Accuracy and memory cost are compared for various models including RK-Net which has basically the same architecture as ODE-Net but to compute gradients, backpropagation is used with Runge-Kutta Integrator as ODESolve. Here $L$ is the number of layers in ResNet and $\tilde{L}$ is the number of function evaluations that the ODE solver requests in a single forward pass, which can be interpreted as an implicit number of layers.

It is found that ODE-Nets and RK-Nets can achieve around the same performance as the ResNet.

### 1.3.2 Error Control in ODE-Nets

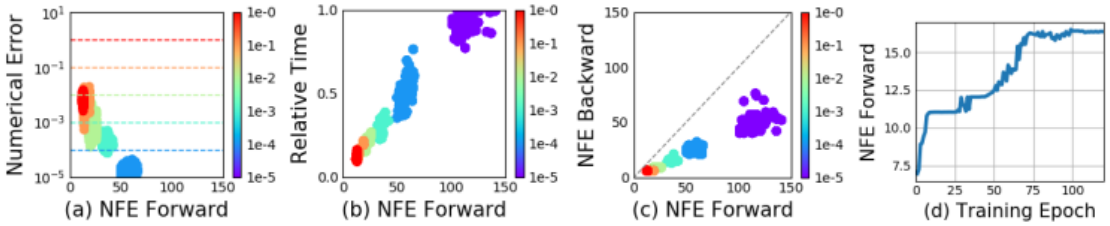


Figure 3: Statistics of a trained ODE-Net. (NFE = number of function evaluations.)

ODE solvers can approximately ensure that the output is within a given tolerance of the true solution. Changing this tolerance changes the behavior of the network.

While $3(a)$ and $3(b)$ are as expected, $3(c)$ shows that the number of evaluations in the backward pass is roughly half of the forward pass. This suggests that the adjoint sensitivity method is not only more memory efficient, but also more computationally efficient than directly backpropagating through the integrator.

# 2 Fractional Differential Equations

Naturally, after reading about neural ordinary differential equations, the next step is to read about the less widely known topic of Fractional calculus and fractional differential equations, which is crucial in order to formulate, if that is possible hopefully, the **Neural Fractional Differential Equations**.

## 2.1 Special Functions

These functions play very important role in the theory of differentiation of arbitrary order and in the theory of fractional differential equations.

### 2.1.1 Gamma function

One of the. basic functions of the fractional calculus is Euler's Gamma function $\Gamma(n)$ which generalizes the factorial $n!$ and allows $n$ to take also

non-integer and even complex values. Gamma function is defined as,

$$\Gamma(z) = \int_0^\infty e^{-t} t^{z-1} dt \quad Re(z) > 0 \tag{8}$$

Here are some easily provable properties of gamma function,

$$\Gamma(z+1) = z\Gamma(z) \quad z \in \mathbb{R} \tag{9}$$

$$\Gamma(n+1) = n! \quad n \in \mathbb{N} \tag{10}$$

$$\Gamma(\frac{1}{2}) = \sqrt{\pi} \tag{11}$$

$$\Gamma(z)\Gamma(1-z) = \frac{\pi}{sin(\pi z)} \tag{12}$$

### 2.1.2 Mittag-Leffler Function

The exponential function $e^z$, plays a very important role in the theory of integer-order differential equations. Its one-parameter generalization, the function which is now denoted as,

$$E_\alpha(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k + 1)} \tag{13}$$

Two-parameter version of Mittag-Leffler function is of more importance,

$$E_{\alpha,\beta}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k + \beta)} \qquad (\alpha > 0, \beta > 0) \tag{14}$$

It follows from the above definition,

$$E_{1,1}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(k+1)} = \sum_{k=0}^{\infty} \frac{z^k}{k!} = e^z \tag{15}$$

$$E_{1,2}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(k+2)} = \sum_{k=0}^{\infty} \frac{z^k}{(k+1)!} = \frac{1}{z} \sum_{k=0}^{\infty} \frac{z^{k+1}}{(k+1)!} = \frac{e^z - 1}{z} \tag{16}$$

$$E_{1,3}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(k+3)} = \sum_{k=0}^{\infty} \frac{z^k}{(k+2)!} = \frac{1}{z^2} \sum_{k=0}^{\infty} \frac{z^{k+2}}{(k+2)!} = \frac{e^z - 1 - z}{z^2} \tag{17}$$

In General,

$$E_{1,m}(z) = \frac{1}{z^{m-1}}\Big(e^z - \sum_{k=0}^{m-2} \frac{z^k}{k!}\Big) \tag{18}$$

Some important particular cases are sine, cosine, hyperbolic cosine and hyperbolic sine functions,

$$E_{2,1}(z^2) = \sum_{k=0}^{\infty} \frac{z^{2k}}{\Gamma(2k+1)} = \sum_{k=0}^{\infty} \frac{z^{2k}}{(2k)!} = cosh(z) \tag{19}$$

$$E_{2,2}(z^2) = \sum_{k=0}^{\infty} \frac{z^{2k}}{\Gamma(2k+2)} = \frac{1}{z}\sum_{k=0}^{\infty} \frac{z^{2k+1}}{(2k+1)!} = \frac{sinh(z)}{z} \tag{20}$$

$$E_{2,1}(-z^2) = cos(z) \tag{21}$$

$$E_{2,2}(-z^2) = \frac{sin(z)}{z} \tag{22}$$

Few more easy to derive results involving Mittag-Leffler functions are its derivative,integral and Laplace Transform,

$$L(z^{\beta-1}E_{\alpha,\beta}(\lambda t^\alpha)) = \int_0^\infty e^{-st}t^{\beta-1}E_{\alpha,\beta}(zt^\alpha)dt = \frac{s^{\alpha-\beta}}{s^\alpha - z} \tag{23}$$

$$\frac{d}{dz}E_{\alpha,\beta}(z) = \frac{1}{\alpha z}E_{\alpha,\beta-1} - \frac{\beta-1}{\alpha z}E_{\alpha,\beta} \tag{24}$$

$$\Big(\frac{d}{dt}\Big)^m \big(t^{\beta-1}E_{m,\beta}(t^m)\big) = t^{\beta-1}E_{m,\beta}(t^m) \quad (m = 1, 2, ...; \beta = 0, 1, ..., m) \tag{25}$$

$$\int_0^z E_{\alpha,\beta}(\lambda t^\alpha)t^{\beta-1}dt = z^\beta E_{\alpha,\beta+1}(\lambda z^\alpha) \quad (\beta > 0) \tag{26}$$

## 2.2 Fractional Derivatives and Integrals

The fractional calculus is a name for the theory of integrals and derivatives of arbitrary order which unify and generalize the notions of integer-order differentiation and $n$-fold integration.

### 2.2.1 Grünwald-Letkinov Fractional Derivatives

First, we will unify the integer-order derivatives with integrals, to begin with, let us consider our same old definition of derivative of $f(t)$,

$$f'(t) = \lim_{h\to 0} \frac{f(t) - f(t-h)}{h} \tag{27}$$

Using this definition multiple times, we get,

$$f''(t) = \lim_{h\to 0} \frac{f(t) - 2f(t-h) + f(t-2h)}{h^2} \tag{28}$$

$$f^{(n)}(t) = \frac{d^n f}{dt^n} = \lim_{h\to 0} \frac{1}{h^n} \sum_{r=0}^{n} (-1)^r \binom{n}{r} f(t-rh), \quad h = \frac{t-a}{n}, n \in \mathbb{N} \tag{29}$$

Generalizing to fractions, we get,

$$f^{(p)}(t) = \lim_{h\to 0} \frac{1}{h^p} \sum_{r=0}^{n} (-1)^r \binom{p}{r} f(t-rh) \tag{30}$$

Generalizing using Gamma function,

$$\binom{p}{r} = \frac{\Gamma(p+1)}{(r)!\Gamma(p-r+1)} \tag{31}$$

We get the **Grünwald-Letkinov Fractional Derivative**,

$${}_aD_t^{(p)} f(t) = f^{(p)}(t) = \lim_{h\to 0} \frac{1}{h^p} \sum_{r=0}^{n} (-1)^r \frac{\Gamma(p+1)}{(r)!\Gamma(p-r+1)} f(t-rh) \tag{32}$$

As an observation, these derivaties are *linear*, replacing $n \to -n$ to generalize this expression to all fractions, using gamma function and the relation,

$$\binom{-n}{r} = (-1)^r \frac{(n+j-1)!}{j!(n-1)!} = (-1)^r \frac{\Gamma(n+j)}{(j)!\Gamma(n)} \tag{33}$$

We obtain,

$$f^{(-p)}(t) = \lim_{h\to 0} h^p \sum_{r=0}^{n} \frac{\Gamma(p+r)}{(r)!\Gamma(p)} f(t-rh) \tag{34}$$

Now, it is clear that negative order differential is basically positive order integral, hence,

$$f^{(-p)}(t) = I^p f(t) = \lim_{h\to 0} h^p \sum_{r=0}^{n} \frac{\Gamma(p+r)}{(r)!\Gamma(p)} f(t-rh) \tag{35}$$

For p=1 :

$$I^1 f(t) = \lim_{h\to 0} h \sum_{r=0}^{n} \frac{\Gamma(1+r)}{(r)!\Gamma(1)} f(t-rh) = \lim_{h\to 0} h \sum_{r=0}^{n} f(t-rh) \tag{36}$$

$$I^1 f(t) = \int_0^{t-a} f(t-\tau)d\tau \tag{37}$$

$$I^1 f(t) = \int_a^{t} f(\tau)d\tau \tag{38}$$

This is indeed the integral of f(t). Equation (35) is called the **Riemann-Liouville fractional integral**.

## 2.3 Methods of solving Linear Fractional Differential Equations

We discuss 2 of many methods to solve fractional differential equations.

### 2.3.1 Laplace transform method

First we write basic Laplace Transform results that we'll need,

$$L({}_0I_x^\alpha f(x)) = s^{-\alpha}F(s) \tag{39}$$

$$L({}_0D_x^\alpha f(x)) = s^\alpha F(s) - \sum_{k=0}^{n-1} s^k {}_0D_x^{\alpha-k-1} f(0), \;\; n-1 < \alpha \leq n \tag{40}$$

$$L(x^{\beta-1}E_{\alpha,\beta}(\lambda x^\alpha)) = \frac{s^{\alpha-\beta}}{s^\alpha - \lambda} \tag{41}$$

Considering the following homogeneous equation, with given initial value,

$${}_0D_x^{1/2} y(x) + ay(x) = 0, \quad x > 0; [{}_0D_x^{-1/2} y(x)]_{x=0} = c \tag{42}$$

Performing Laplace transform both sides,

$$L({}_0D_x^{1/2} y(x)) + aL(y(x)) = 0 \tag{43}$$

$$\tag{44}$$

From (40),

$$s^{1/2}Y(s) - {}_0D_x^{-1/2} y(0) + Y(s) = 0 \tag{45}$$

$$\implies Y(s) = \frac{c}{a + s^{1/2}} \tag{46}$$

Using (41), we get the Inverse Laplace transform of $Y(s)$,

$$y(x) = Cx^{-1/2}E_{1/2,1/2}(-ax^{1/2}) \tag{47}$$

## 2.4 Power Series Method

Considering a simple initial-value problem,

$$ {}_0D_x^{\alpha} y(x) = f(x), \quad y(0) = 0 \tag{48} $$

Performing the following substitutions,

$$ f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} x^n \tag{49} $$

$$ y(n) = x^{\alpha} \sum_{n=0}^{\infty} c_n x^n \tag{50} $$

(48) now becomes,

$$ {}_0D_x^{\alpha} \left( \sum_{n=0}^{\infty} c_n x^n \right) = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} x^n \tag{51} $$

$$ \sum_{n=0}^{\infty} c_n [{}_0D_x^{\alpha}(x^n)] = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} x^n \tag{52} $$

$$ \tag{53} $$

Using the results,

$$ {}_0D_x^{\alpha}(x^n) = \frac{\Gamma(n+1)}{\Gamma(n-\alpha+1)} x^{n-\alpha} \tag{54} $$

$$ \Gamma(n+1) = n! \tag{55} $$

We get,

$$ \sum_{n=0}^{\infty} c_n \frac{\Gamma(n+\alpha+1)}{\Gamma(n+\alpha-\alpha+1)} x^{n+\alpha-\alpha} = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} x^n \tag{56} $$

$$ \sum_{n=0}^{\infty} c_n \frac{\Gamma(n+\alpha+1)}{n!} x^{\alpha} = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} x^n \tag{57} $$

$$ \implies c_n = \frac{f^{(n)}(0)}{\Gamma(n+\alpha+1)} \tag{58} $$

# References


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