Using a Neural Network Trained Only on Integer Order Systems to Identify Fractional Order Dynamics in Large Scale Systems

Bill Goodwine[1] and Tan Chen[2]

***Abstract*— This paper presents a standard artificial neural network that identifies the order of the dynamics of a unit step response. The system is trained on only first and second order systems, yet identifies fractional order responses with a high degree of accuracy. The details of the structure of the neural network, the training method and the training sets, as well as statistics describing the accuracy of the fractional predictions are presented. Also using the neural network to identify fractional dynamics for a large scale networked system from the authors' prior work is presented as further validation and a demonstration of the applicability of the results. This demonstrates the potential for practicing engineers to use similar machine learning tools trained on "standard" systems with the ability to distinguish when features such as fractional order dynamics are significant and warrant deeper consideration for the design or control of such a system.**

I. INTRODUCTION

Fractional calculus and fractional order dynamics are increasingly important in modern engineered systems. Unlike integer order derivatives, fractional order derivatives, and hence the dynamics that depend on them, are *nonlocal*. As such, many modern, large scale engineered systems may exhibit fractional order dynamics and responses because interactions among various components in the system may be significantly displaced in time or space. In instances where significant fractional order dynamics are present, control algorithms which directly address the fractional nature of the system may be superior. Therefore, tools to readily identify if significant fractional order dynamics are present are needed.

There is a vast literature on fractional calculus. Some textbooks include [1]–[3]. Fractional-order control is considered in many topical areas, but particularly relevant to this paper is fractional order controls, such as in [4], [5]. An excellent review article illustrating the very broad range of applications of fractional calculus and control in science and engineering is [6].

Our main interest are identifying cases where fractional order models may provide useful "reduced order" models for large scale systems [7], [8] and for exact models for many large scale systems [9]–[13]. A fractional order system is arguably infinite order, but a fractional order representation is more concise than, for example, an extremely high order system like those considered in the preceding references. While this paper does not build upon it, our closest publication to this would be [14] where we created a symmetric neural network with a sequential set of identical layers. When it was trained on first derivatives of functions, the middle layer could represent the half derivative.

There are many different definitions of the fractional derivative. As will be outlined in the next section, a common feature of these is replacing factorial functions appearing in many integer-order representations of the derivative with gamma functions. The Riemann-Liouville, Caputo and the Grünwald-Letnikov definitions are perhaps the most common examples of fractional derivative definitions, and the reader is referred to the references [15]–[18] for descriptions and definitions of each. Because of the python library we utilize for this paper, the definition used herein is the Caputo fractional derivative.

II. Fractional Calculus

This section gives a very brief overview of only the fractional calclulus topics necessary to implement the methods in this paper and is therefore incomplete. The interested reader is referred to the references above for a complete exposition.

The obvious task for fractional calculus to to define derivatives "in between" the usual integer order derivatives, *e.g.*, the one half derivative of $x(t)$. Consider the first and second order backwards finite difference equations for $\Delta t \ll 1$

$$\frac{\mathrm{d}x}{\mathrm{d}t}(t) \approx \frac{x(t) - x(t - \Delta t)}{\Delta t}$$

and

$$\frac{\mathrm{d}^2 x}{\mathrm{d}t^2}(t) \approx \frac{x(t) - 2x(t - \Delta t) + x(t - 2\Delta t)}{(\Delta t)^2}.$$

As is well known, from these it is clear that the formula for an arbitrary positive integer order derivative is

$$\frac{\mathrm{d}^n x}{\mathrm{d}t^n}(t) \approx \frac{\sum_{i=0}^{n} (-1)^i \left(\frac{n!}{i!(n-i)!}\right) x(t - i\Delta t)}{(\Delta t)^n}, \qquad (1)$$

where the fraction with the factorials in the numerator and denominator is the binomial coefficient. Note that the number of terms in the sum is one more than the

[1]Bill Goodwine is with the Department of Aerospace & Mechanical Engineering, University of Notre Dame, Notre Dame, IN 46556, USA billgoodwine@nd.edu

[2]Tan Chen is with the Department of Electrical and Computer Engineering, Michigan Technological University, Houghton, MI 49931, USA tanchen@mtu.edu

order of the derivative. The only terms in Equation 1 that must be integers are the factorials. Because the gamma function can be considered a generalization of the factorial because $n! = \Gamma(n+1)$ for integer values of n, a generalized equation results with

$$\frac{\mathrm{d}^\alpha x}{dt^\alpha}(t) \approx \frac{\sum_{i=0}^{\lceil \frac{t}{\Delta t} \rceil} (-1)^i \left(\frac{\Gamma(\alpha+1)}{i!\Gamma(\alpha-i+1)} \right) x(t - i\Delta t)}{(\Delta t)^\alpha}. \quad (2)$$

The generalized binomial coefficient in Equation 2 is nonzero for all values of i (except when α is an integer), and hence the sum will include times spanning all of history. In the controls context assuming zero initial conditions and zero values for all time prior to zero, the sum will include terms at all time steps between 0 and the current t. This highlights an important distinction between integer order derivatives and fractional order derivatives, which is that the latter are *nonlocal*.

The method to compute the fractional step responses that the neural network will identify was computed by a different method than what is expressed in Equation 2, but is similar in spirit. Specifically, we use the `numfracpy` python library to numerically compute fractional step responses, and from its documentation[1] the step response is computed using a fractional order Adams predictor corrector method given by

$$x_{n+1}^P = \sum_{j=0}^{m-1} \frac{t_{n+1}^j}{j!} x_0^j + (\Delta t)^\alpha \sum_{j=0}^{n} b_{j,n+1} f(t_j, x_j)$$

$$x_{n+1} = \sum_{j=0}^{m-1} \frac{t_{n+1}^j}{j!} x_0^j + \sum_{j=0}^{n} \left[a_{j,n+1} f(t_j, x_j) + a_{n+1,n+1} f\left(t_{n+1}, x_{n+1}^P\right) \right]$$

where

$$b_{j,n+1} = \frac{1}{\Gamma(\alpha+1)} \left[(n-j+1)^\alpha - (n-j)^\alpha \right]$$

and

$$\alpha_{0,n+1} = \frac{(\Delta t)^\alpha \left[n^{\alpha+1} - (n-\alpha)(n+1)^\alpha \right]}{\Gamma(\alpha+2)},$$

$$a_{j,n+1} = \frac{(\Delta t)^\alpha}{\Gamma(\alpha+2)} \left[(n-j+2)^{\alpha+1} - 2(n-j+1)^{\alpha+1} + (n-j)^{\alpha+1} \right]$$

for $1 \leq j \leq n$ and

$$\alpha_{n+1,n+1} = \frac{(\Delta t)^\alpha}{\Gamma(\alpha+2)}.$$

The summations in the expressions for x_{n+1} are the parts of the expressions that include the nonlocal terms.

[1]http://tinyurl.com/yyytpv8d

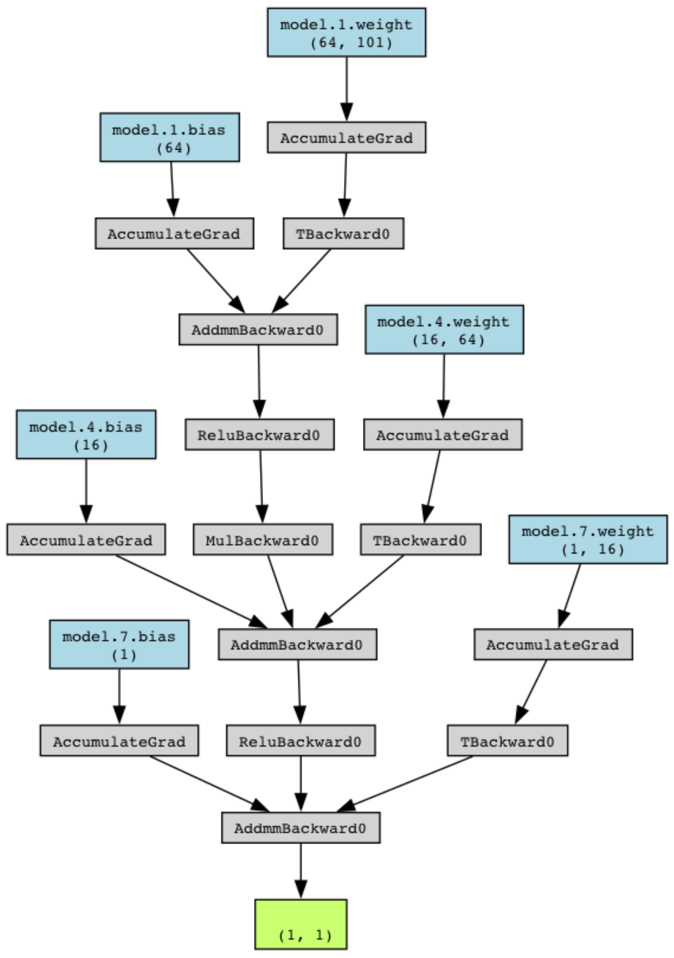

Fig. 1. The neural network.

III. Neural Network and Training

The neural network used in this paper is illustrated in Figure 1. There is an input layer with 101 nodes, a hidden layer with 64 nodes, a subsequent hidden layer with 16 nodes, another hidden layer with 16 nodes, and an output layer with 1 node. Each hidden layer has the ReLU activation function. The values input to the network is the unit step response of a linear system in the time range of $0 \leq t \leq 10$ discretized into time steps of $\Delta t = 0.1$ [s], which gives 101 input nodes. The single output of the network is the order of the system that produced the input step response.

The network was implemented in python using the `torch` library and `pytorch_lightning` tools. This network is not trained as a classifier because we want it to be able to generalize first and second order systems to fractional orders between them, so the loss function is the mean squared error function, `mse_loss()`, and the optimization method adopted was Adam optimizer, `torch.optim.Adam()` with a learning rate of 0.001. A branch of our github repository that should repeatably replicate the results presented in this paper is at **ZZZZZZ**.

A. Integer Order Training, Validation and Testing Data

An individual element of the training, validation and training sets is the step response for a first or second order system (not fractional order). The manner in which they are generated is:

- Select a value from a uniform random distribution between 0 and 1, and if the value is less than 0.5, then the step response will be for a second order system, and if not, then it will be for a first order system.
- Select two numbers, c_1 and c_2 from a uniform distribution with values between 0.01 and 4.
  - If the response is for a first order system, then the transfer function is
  $$G(s) = \frac{c_2}{c_1 s + c_2}.$$
  - If the response is for a second order system, then the selected transfer function is
  $$G(s) = \frac{c_2}{c_1 s^2 + c_2}.$$
- The unit step response from 0 to 10 seconds for the transfer function is generated using the `control.step_response()` function from the python control system library. The number of time steps in the response is determined by the step response function. It is then sampled every 0.1 second so that the length of the response vector is 101.

Using this method we generate a set of 100,000 first or second order step responses with approximately the same number of first and second order responses. The training set is split into three subsets: 60,000 training elements, 20,000 validation elements and 20,000 testing elements. For training, the training set is shuffled at the beginning of each epoch and the optimization method is applied to change the weights in the network. At the end of the epoch, the network is run on the validation set to compute an error for data points the network was not trained on. Evidence of overtraining would be if the validation set error decreases and then increases. At the end of all the training, the error for the testing set is computed.

Figure 2 illustrates the error on the validation set for 10 training runs for the network versus epoch. It appears that if the validation error increases, it tends to start to do so around 1000 epochs; otherwise it tends to stop changing around 1000 epochs (there seems to be one exception). As such, we will fix the number of training epochs at 1000. Additionally, various alternative configurations to the network described above, including more hidden nodes and different activation functions were investigated, and the configuration presented above generally gave the best results.

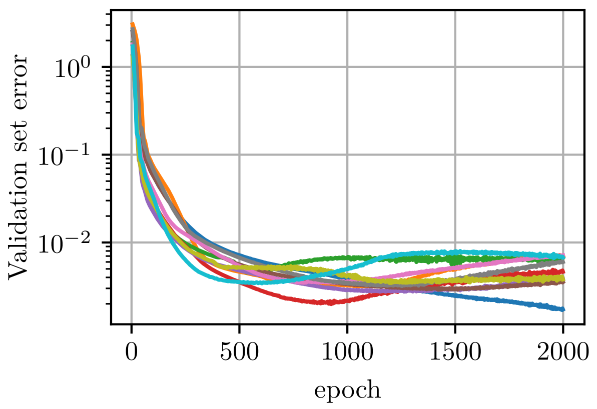

Fig. 2. Neural network output error on validation (not training) set versus training epoch.

IV. Using the Integer Trained Network on Fractional Order Step Responses

Now we give a fractional order step response to the trained neural network to see if it can generalize the training on first and second order transfer functions to fractional order step responses. As indicated above, we use the `numfracpy` python library to numerically compute the fractional order step response for time from 0 to 10. The solution is numerically computed with a time step of $\Delta t = 0.01$, but the input to the nextwork is 101 nodes, so only every 10th element of the numerical solution is used.

We tested the network on 1000 fractional order step responses to transfer functions of the form

$$X(s) = \left(\frac{k}{s^\alpha + k}\right)\left(\frac{1}{s}\right),$$

or in the time domain

$$\frac{\mathrm{d}^\alpha x}{\mathrm{d}t^\alpha}(t) + kx(t) = k \qquad (3)$$

with zero initial conditions.

The 1000 responses were each generated and tested as follows:

- Randomly select an order between 1 and 2 from a uniform distribution.
- Randomly select a k between 5 and 9 from a uniform distribution. This range of values was selected to produce responses with a period of oscillation similar to the natural frequencies in the training set.
- Numerically compute the step response to Equation 3 using the `FODE()` function from the `numfracpy` library.

more text

V. Conclusions

A conclusion section is not required. Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions.

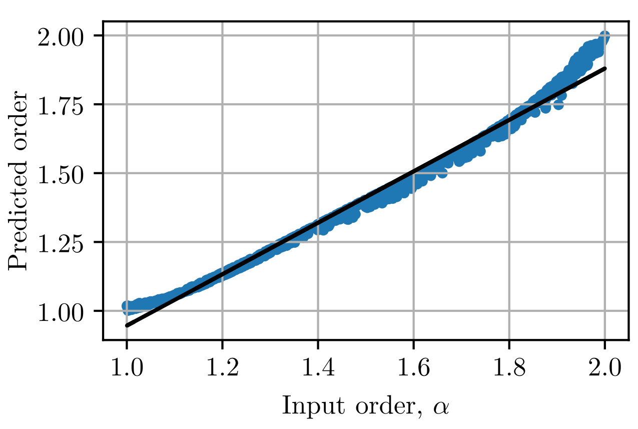

Fig. 3. Comparison of actual fractional orders and predicted orders.

References

[1] M. D. Ortigueira, *Fractional Calculus for Scientists and Engineers*, ser. Lecture Notes in Electrical Engineering. Netherlands: Springer Netherlands, 2011, vol. 84.

[2] K. Oldham and J. Spanier, *The Fractional Calculus Theory and Applications of Differentiation and Integration to Arbitrary Order.* Elsevier Science, 1974.

[3] A. Oustaloup, *La d´erivation non enti‘ere.* Hermes, 1995.

[4] D. Valério and J. S. de Costa, *An Introduction to Fractional Control.* London, United Kingdom: Institution of Engineering and Technology, 2013.

[5] Dingyu Xue, Chunna Zhao, and YangQuan Chen, "Fractional order PID control of a DC-motor with elastic shaft: a case study," in *2006 American Control Conference*, 2006, pp. 3182–3187.

[6] H. Sun, Y. Zhang, D. Baleanu, W. Chen, and Y. Chen, "A new collection of real world applications of fractional calculus in science and engineering," *Communications in Nonlinear Science and Numerical Simulation*, vol. 64, pp. 213–231, 2018.

[7] B. Goodwine, "Fractional-order dynamics in large scale control systems," in *2023 31st Mediterranean Conference on Control and Automation (MED)*, 2023, pp. 747–752.

[8] ——, "Factors influencing fractional-order dynamics in large, scale-free robotics swarms," in *2023 27th International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2023, pp. 382–387.

[9] B. Goodwine, "Modeling a multi-robot system with fractional-order differential equations," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 1763–1768.

[10] K. Leyden and B. Goodwine, "Using fractional-order differential equations for health monitoring of a system of cooperating robots," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 366–371.

[11] K. Leyden, M. Sen, and B. Goodwine, "Large and infinite mass–spring–damper networks," *Journal of Dynamic Systems, Measurement, and Control*, vol. 141, no. 6, Feb 2019, 061005. [Online]. Available: https://doi.org/10.1115/1.4042466

[12] X. Ni and B. Goodwine, "Frequency Response and Transfer Functions of Large Self-Similar Networks," *Journal of Dynamic Systems, Measurement, and Control*, vol. 144, no. 8, 06 2022, 081007. [Online]. Available: https://doi.org/10.1115/1.4054645

[13] ——, "Damage modeling and detection for a tree network using fractional-order calculus," *Nonlinear Dynamics*, vol. 101, pp. 875–891, 2020.

[14] T. Chen and B. Goodwine, "A symmetric neural network to compute fractional derivatives by training with integer

derivatives," in *2022 IEEE/SICE International Symposium on System Integration (SII)*, 2022, pp. 291–296.

[15] J. T. Machado, V. Kiryakova, and F. Mainardi, "Recent history of fractional calculus," *Communications in Nonlinear Science and Numerical Simulation*, vol. 16, no. 3, pp. 1140 – 1153, 2011.

[16] M. Ortigueira, "An introduction to the fractional continuous-time linear systems: the 21st century systems," *Circuits and Systems Magazine, IEEE*, vol. 8, no. 3, pp. 19–26, 2008.

[17] M. D. Ortigueira, *Fractional Calculus for Scientists and Engineers*, ser. Lecture Notes in Electrical Engineering. Springer, 2011, vol. 84.

[18] S. Das, *Functional Fractional Calculus.* Springer Science & Business Media, 2011.