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

Bill Goodwine[1] and Tan Chen[2]

***Abstract*— This paper presents a standard feed forward artificial neural network that identifies the order of the dynamics of a unit step response. The main contribution of this paper is the demonstration that a system that is trained on only integer order, first and second order systems, can identify fractional order responses with a high degree of accuracy. The details of the design of 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 usually *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. Furthermore, independent of control, having a good mathematical model for a system is useful in analysis of system performance, design, and design optimization. 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].

[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`

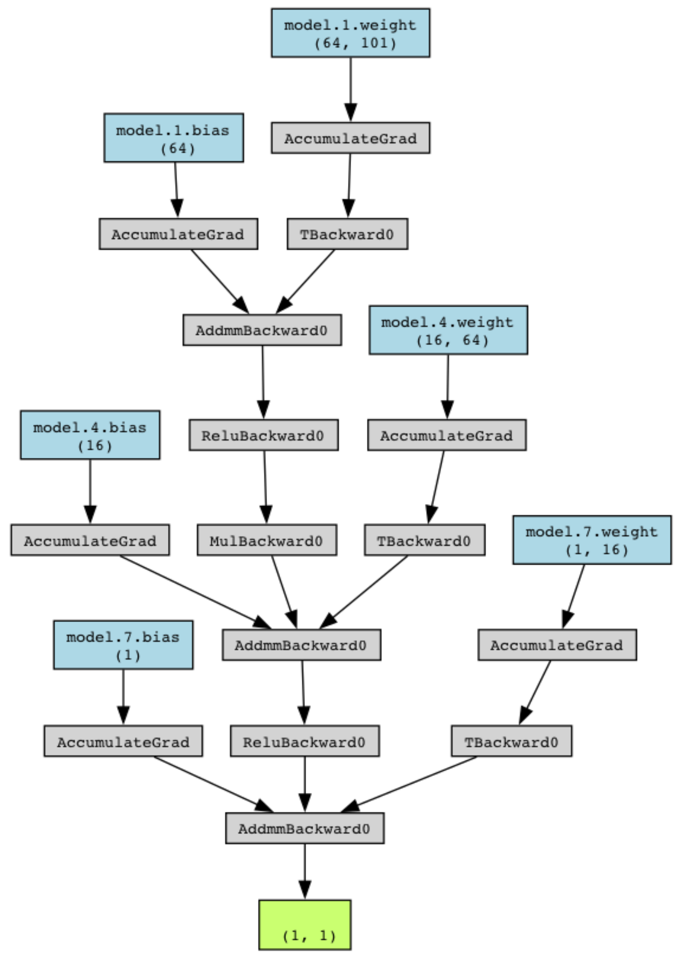

Fig. 1. The neural network.

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.

Machine learning in general and neural networks in particular have a long history. One of the earliest works related to the feed forward neural network, the type used in this paper, is from 1960 by Rosenblatt [19]. In the 1970s, advanced in training by the back propagation method were developed [20] and were further advanced in the 1980s [21]. Additionally in the 1980s theoretical advances such as the universal approximation theorem [22] were developed. These practical advances buttressed with theoretical support helped advance the cause for usefulness of artificial neural networks. More recently, the confluence of ubiquitous data and advanced in computer processing power enabled many applications in the beginning of the 21st century has resulted in the rapid deployment throughout many aspects of modern life.

Because robotics and control systems are at the interface of computational intelligence and the physical world, there have been many applications of machine learning and neural networks in this realm. Space limitations prohibit a comprehensive outline of the literature. However, several recent survey papers include [23]–[25] highlighting many application areas such as assembly and object handling in manufacturing, human robot interaction, object recognition and other vision applications, locomotion and path planning, transfer learning, localization, grasp detection, motion planning, whole body planning, sensor fusion, control, and many others.

The rest of this paper is organized as follows. Section II provides a limited introduction to fractional calculus, including the concepts needed for this paper. Section III presents the details of the artificial neural network used and the integer order training data. Section IV presents the results when the neural network is used to identify fractional order dynamics. Section V further validates the results by applying the network to identify fractional order dynamics in a large scale networked system from the literature. Finally, Section VI presents the conclusions and an outline of ongoing and future work.

II. FRACTIONAL CALCULUS

This section gives a very brief overview of only the fractional calculus 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 is to define derivatives "in between" the usual integer order derivatives, *e.g.*,

$$x(t) = \frac{\mathrm{d}^0 x}{\mathrm{d}t^0}(t), \quad \frac{\mathrm{d}^{1/2} x}{\mathrm{d}t^{1/2}}(t), \quad \frac{\mathrm{d}x}{\mathrm{d}t}(t), \quad \frac{\mathrm{d}^{5/4} x}{\mathrm{d}t^{5/4}}(t), \ldots .$$

For one approach to generalizing the derivative to real, possibly noninteger, orders, 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

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

From these two equations it is clear that the formula for an arbitrary positive integer order derivative is

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

where the fraction that is the coefficient of $x(t-i\Delta t)$ 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 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

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

where the symbol for the order of the derivative was changed from n to α to connote that it no longer must be an integer. If, instead of $\Delta t \ll 0$ the limit as $\Delta t \to 0$ is used, this equation becomes the Grünwald-Letnikov derivative (see the references cited in the introduction for a complete exposition).

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*. It also highlights the fact that computing fractional derivatives is computationally expensive compared to integer order.

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

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

is computed using a fractional order Adams predictor corrector method given by

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

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

where

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

and

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

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

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.

III. NEURAL NETWORK, TRAINING DATA SET, AND TRAINING

A. *Selection of the Structure of the Neural Network*

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]. That value of Δt was selected to give approximately 20 points in one typical period of oscillation in the dataset, which then specifies the number of 101 input nodes. The selection of the activation functions and number of hidden nodes and layers is presented in the next section. The single output of the network is the order of the system that produced the input step response. There are approximately 7600 parameters in the feedforward neural network that are adaptable for training.

The network was implemented in python using the `torch` library and `pytorch_lightning` tools. This network is not trained as a pure 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 has fixed random seeds that should repeatably replicate the results presented in this paper is at [26].

B. *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. This range was selected to give a frequency range of training data between nearly zero and 400.
  - 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.

C. *Training method*

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. Using a training set of that size corresponds to approximately 8 training points for each parameter in the network. For training, the training set is shuffled at the beginning of each epoch and the optimization method is applied to change the parameters 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 over training 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.

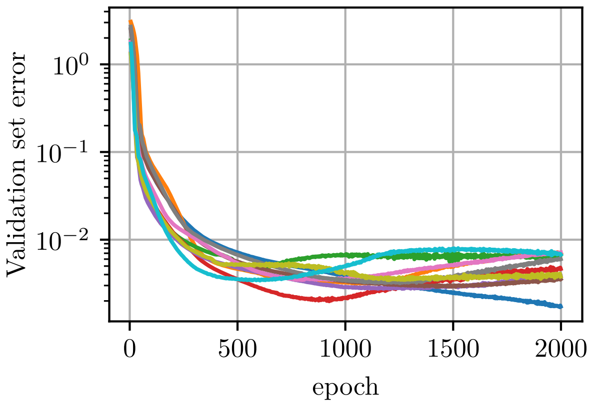

Fig. 2. Neural network output error on validation (not training) set versus training epoch.

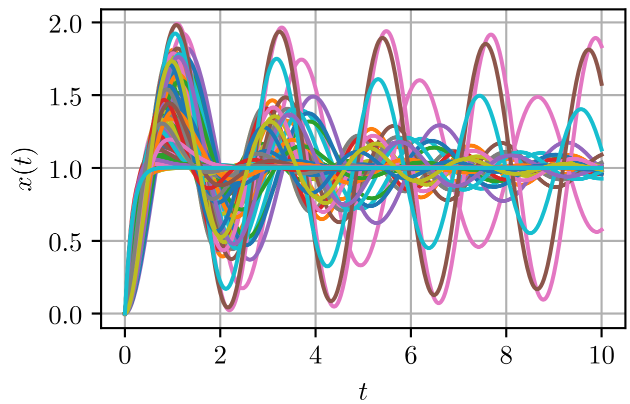

Fig. 3. First 100 fractional order step responses generated and used for the neural network.

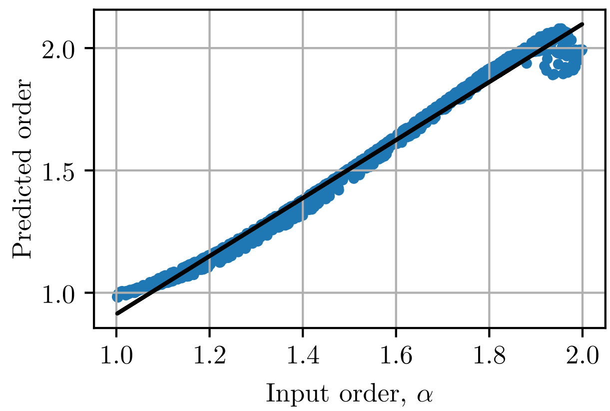

Fig. 4. Comparison of actual fractional orders and predicted orders.

IV. USING THE INTEGER TRAINED NETWORK ON FRACTIONAL ORDER STEP RESPONSES

Now we apply a fractional order step response to the input layer of 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 network is 101 nodes, so only every 10th element of the numerical solution is used.

A. Generating the test set

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

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

where α is the fractional order of the derivative, or in the time domain

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

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.

B. Neural network identifying the order of the test set

Each of the 1000 fractional order step responses was interpolated or sampled into time steps of $\Delta t = 0.1$, and those points were applied to the input layer of the neural network trained on the integer order data. Figure 4 is the main result in this paper. The results that are illustrated in that figure illustrate an excellent match. Each blue dot in the plot corresponds to one element of the fractional test set. Perfect matching would result in the data forming a straight line from the point $(1, 1)$ to $(2, 2)$. The mean square error for the data is 0.00121 with $R^2 = 0.9897$.

Again, we emphasize that the neural network was only trained at 1 and 2, so all of the points along the diagonal line between those values can reasonably be interpreted as a result of the neural network generalizing the definition of the derivative. The largest deviation from perfect generalization is near the two endpoints. A speculative, but plausable, explanation of those is that because those points are near the training points, the network will tend to over predict orders of 1 and 2.

In designing the neural network, we tested the efficacy of the network on a wide range of configurations; specifically, with the number of nodes in the first hidden layer ranging from 32 to 128 and the number of nodes in the second hidden layer ranging from 16 to 64. We also tested ReLU and sigmoid activation functions. As a measure of efficacy in predicting the fractional order, the R^2 ranged from 0.989 for the network configuration presented to 0.95 for sigmoid activation functions with the same configuration of hidden nodes to 0.92 with a higher number of hidden nodes.

V. Application to Scale Free Network with Fractional Order Dynamics

In this section we use this network on a system from one of our prior publications. In [7], [8] we studied the effects of various parameters in large, scale free networks on the existence of fractional order dynamics. To further validate the generalizability of the results for the neural network, we apply it this network to the response of a large, scale free network with fractional order dynamics.

Specifically, we consider a network with 2000 nodes. Each node has a mass of 1 and is connected to various other nodes with either a spring or damper, with coefficients $k = 2501$ and $b = 150$. The details of the manner in which the network is constructed and justification of the selection of the various parameter values are contained in the references. The step response of the system when one of the masses has a non-zero initial condition is computed, and the response of one of the nodes is observed. The network used for this validation is illustrated in Figure 5 and the response of node 1011 when node 100 was displaced by 1 was determined by numerically solving the system of 4000 differential equations and is illustrated in Figure 6.

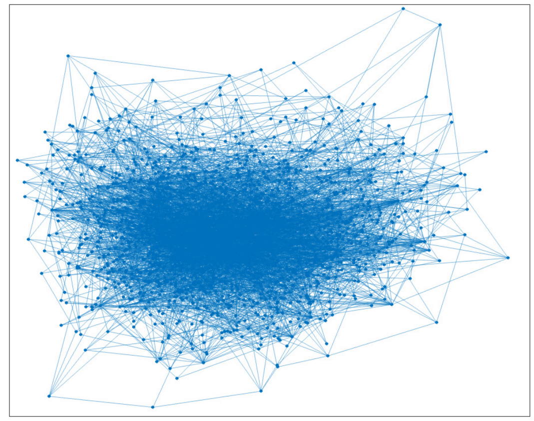

Fig. 5. Large scale free network used for validation.

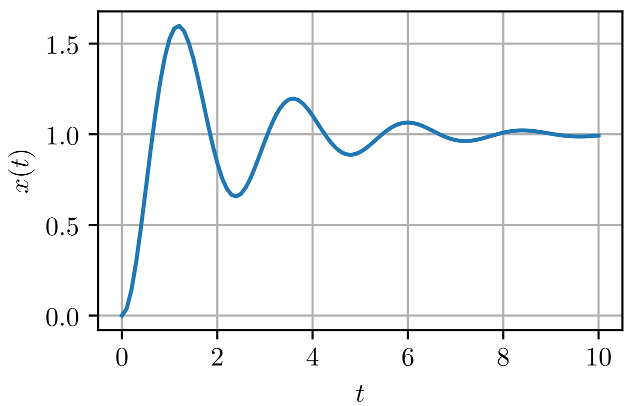

Fig. 6. Response of scale free network.

Because it is a mechanical system with 2000 nodes, the overall system has extremely high order dynamics. The purpose of the research outlined in the references is to determine reduced order models and determine when a fractional order "reduced order" model better matches the response than a standard second order model. Using two different optimization methods (Matlab `patternsearch` (deterministic) and `particleswarm` (stochastic)) the best fit of a fractional order unit step response and second order step response were determined.

We then compare the order determined by the optimization method, which is very slow, to the order predicted by the neural network, which is very fast. The predicted order by the neural network was 1.782. The predicted order by the two optimization methods was 1.784, showing nearly exact agreement.

VI. Conclusions and Future Work

This paper presents a neural network that is trained on integer order step responses and predicts with surprising accuracy the order of fractional order step responses. Figure 4 illustrates the accuracy of the fractional order predictions with nearly every fractional step input identified with an order that places it very close to the exact line that would run from the point $(1, 1)$ to $(2, 2)$.

While the network predicts fractional orders very well for fractional orders *between* the trained orders of 1 and 2, it does a poor job extrapolating outside of the range $[1, 2]$, as is illustrated in Figure 7 where the range of fractional values was extended only by 5% in at each end, to $[0.95, 2.05]$. Clearly it is particularly inaccurate for orders above 2. Other than being better at interpolation than at extrapolation, the reason for this is currently not clear and part of our current efforts. Clearly, values very far outside the training set will not be modeled by the network very well, but this network is striking to us in this regard.

There are several avenues for continued research. Currently the network only identifies first order, second order and fractional orders between one and two step

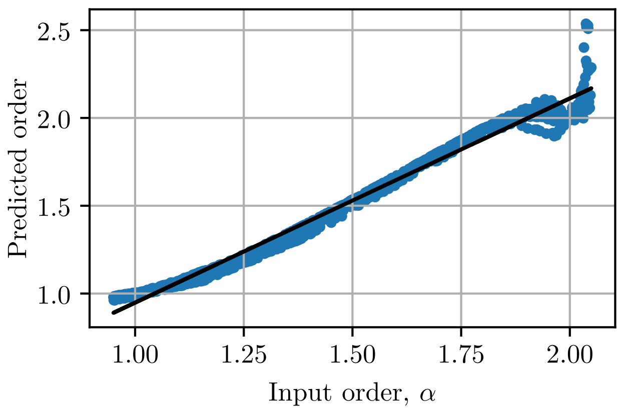

Fig. 7. Comparison of actual fractional orders and predicted orders for $\alpha \in [0.95, 2.05]$.

responses. While it does it remarkably accurately, it only applies to equations of the type represented by Equation 3. Many fractional order systems may have multiple fractional derivative terms and also orders outside the range of $[1, 2]$. With regard to the latter limitation, we have not yet tried the rather obvious approach of adding zeroth order step responses to the system, which would themselves be step functions. These would seem to lack diversity of dynamics to warrant having a third of the training set be them and careful design of the training set and approach would be needed.

This network simply identified fractional orders, albeit very accurately. A much more challenging problem, stemming from our large scale network work, is to distinguish fractional order responses with second order responses with complex roots. In many of our networks considered in [7], [8], the difference between the accuracy of a fractional order model and second order model was fairly small. In such cases a neural network would be tasked with identifying much more subtle differences between integer and fractional order responses. Ongoing efforts are directed toward developing an artificial neural network to do this and evaluating its efficacy.

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.

[19] F. Rosenblatt, "Perceptron simulation experiments," *Proceedings of the IRE*, vol. 48, no. 3, pp. 301–309, 1960.

[20] T. expansion of the accumulated rounding error, "Seppo linnainmaa," *BIT*, p. 146–160, 1976.

[21] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *System Modeling and Optimization*, R. F. Drenick and F. Kozin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, pp. 762–770.

[22] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[23] F. Semeraro, A. Griffiths, and A. Cangelosi, "Human–robot collaboration and machine learning: A systematic review of recent research. robotics and computer-integrated manufacturing," vol. 79, 2023, 102432.

[24] A. I. Károly, P. Galambos, J. Kuti, and I. J. Rudas, "Deep learning in robotics: Survey on model structures and training strategies," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 1, pp. 266–279, 2021.

[25] Q. Bai, S. Li, J. Yang, Q. Song, Z. Li, and X. Zhang, "Object detection recognition and robot grasping based on machine learning: A survey," *IEEE Access*, vol. 8, pp. 181 855–181 879, 2020.

[26] B. Goodwine and T. Chen, "Integer trained neural network to identify fractional order dynamics." [Online]. Available: https://github.com/jwgoodwine/med2024.git