Application of Neural Networks for Financial Options Valuation using Fractional Order Gradient Descent

Omar Bairan
obairan@gmail.com

July 10, 2025

Abstract

We propose a framework for training feed-forward neural networks to price stock options, employing *fractional gradient descent*—an optimization method based on the fractional-order Caputo derivative. After reviewing the application of neural-network for the valuation of stock options and fractional calculus for optimization, we derive update rules using the Caputo derivative of order $\alpha \in (0, 1]$ and apply them on synthetic Black–Scholes data. Numerical experiments show that fractional-order gradient descent ($\alpha < 1$) can be applied to train artificial neural networks and may improve generalization error compared to classical ($\alpha = 1$) gradient descent when using the trained networks on data not previously used for the training process.

Contents

1 Introduction

Artificial neural networks (NNs) have been successfully applied to option pricing problems. For example Hutchinson *et al.* [1] demonstrated the ability of artificial neural networks to learn the Black–Scholes pricing function. Recent advances in deep learning [2] have further improved artificial neural networks architectures and training methods. Also, more recent research [3] has advanced the application of artificial neural networks for pricing financial instruments and extended their application to model the volatility surface of options.

We can consider fractional calculus as a natural extension of classical calculus that allows one to work with derivatives and integrals of any order. In other words, despite its name, this discipline also covers differentiation and integration of real or complex order [4].

In parallel, fractional calculus has emerged as a tool to enhance optimization algorithms via *fractional* (non-integer order) derivatives [5].

In this work, we consider the convergence of these developments by training option-pricing NNs using *fractional gradient descent* (FGD), in which the weight update process for the artificial neural network uses the Caputo derivative of order $\alpha \in (0, 1]$.

As we will show in the experiments that follow, employing a fractional-order gradient descent inherently injects a smoothing, memory-based regularization into the training dynamics of neural networks. By weighting past non-integer gradient information, the optimizer effectively "remembers" a longer history of parameter updates, tempering abrupt weight oscillations and discouraging overfitting to idiosyncrasies in the training set. This long–memory effect behaves similarly to a built-in low-pass filter on the loss landscape, damping high-frequency noise and guiding the network toward broader, flatter minima that generalize more reliably.

In the context of option pricing—where market data are noisy, sparse in extreme regions, and subject to regime shifts—this fractional regularization may translate into markedly improved stability and predictive accuracy when valuing contracts under market conditions not seen during training.

Even though fractional gradient descent may present slower convergence in the training process, this method may yield a pricing engine that is both more robust and more precise on out-of-sample option portfolios.

2 Background

2.1 Neural Networks for Option Pricing

Artificial neural networks (NNs) have emerged as powerful nonparametric approximators, capable of learning complex mappings from market features to option prices without explicit model assumptions. By the universal approximation theorem [6, 7], a sufficiently wide feed-forward network can approximate any continuous function on a compact domain, making it an attractive tool for capturing the nonlinear Black–Scholes pricing map and beyond.

Let

$$x = \big(S,\ K,\ \tau,\ r,\ q,\ \sigma\big) \ = \ (x_1, \dots, x_5) \in [0, \infty)$$

denote the standardized feature vector comprising the price of the underlying asset S, the strike price K, time-to-maturity τ, risk-free interest rate r, continuous dividend yield q, and volatility proxy σ. A fully-connected network with L hidden layers computes

$$\begin{aligned} h^{(0)} &= x, \\ h^{(n)} &= \phi_n\big(W^{(n)}\, h^{(n-1)} + b^{(n)}\big), \quad n = 1, \dots, L, \\ \widehat{C}(x; \mathbf{w}) &= W^{(L+1)}\, h^{(L)} + b^{(L+1)}, \end{aligned}$$

where each $W^{(n)} \in \mathbb{R}^{d_n \times d_{n-1}}$ and $b^{(n)} \in \mathbb{R}^{d_n}$ are the weights and biases, ϕ_n are nonlinear activations (e.g. ReLU or ELU), and $\mathbf{w} = \{W^{(n)}, b^{(n)}\}_{n=1}^{L+1}$ denotes the collection of all parameters.

To train the network, we generate a synthetic dataset of for European Call Options

$$\{(x_i,\, C_{\mathrm{BS}}(x_i))\}_{i=1}^{N}$$

We then compute the ground-truth price C_{BS} via the closed-form Black–Scholes formula [8,9].

Then, we minimize the empirical risk as the loss function:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \big(\widehat{C}(x_i; \mathbf{w}) \;-\; C_{\mathrm{BS}}(x_i)\big)^2, \tag{1}$$

where $\widehat{C}(x_i; \mathbf{w})$ is the price of the call option obtained by the artificial neural network and $C_{\mathrm{BS}}(x_i)$ is the price of the call option obtained from the Black-Scholes model. We then update the weights in the artificial neural network $\mathbf{w}$ iteratively to optimize the loss function.

Although there are faster gradient-based optimizers, such as stochastic gradient descent of Adam [10], our current objective is not training speed, but obtaining more robust results. For the purpose of this work we will compare the results obtained from the training process using gradient descent and fractional gradient descent, with the Caputo fractional derivative.

Early work by Hutchinson et al. [1] showcased the feasibility of NN-based option pricing, while more recent studies [3,11] have extended neural network approximations to rough volatility and model calibration settings. Our approach builds upon these foundations by integrating fractional-order optimization into the training loop, endowing the optimization process for the network with enhanced stability and possible generalization properties.

2.2 Caputo Fractional Derivative

Let $\alpha > 0$, a, $t \in \mathbb{R}$ with $a < t$. We define the Caputo fractional derivative of order α, centered at a, as in [4]:

$${}^{C}D_a^{\alpha+} f(t) := \begin{cases} \dfrac{1}{\Gamma(m-\alpha)} \displaystyle\int_a^t \frac{f^{(m)}(\tau)}{(t-\tau)^{\alpha+1-m}}\, \mathrm{d}\tau, & \alpha \notin \mathbb{N}, \\ \dfrac{\mathrm{d}^m}{\mathrm{d}t^m} f(t), & \alpha = m \in \mathbb{N} \cup \{0\}, \end{cases} \tag{2}$$

where $\Gamma(\cdot)$ denotes the Gamma function. Note that when α is an integer, the Caputo derivative coincides with the standard classical derivative.

Unlike the Riemann–Liouville derivative, the Caputo definition places the derivative $f^{(m)}(\tau)$ within the integral kernel rather than the function $f(\tau)$ itself. This distinction has significant consequences for prescribing initial conditions in fractional differential equations. Moreover, it is important to observe that, in contrast to the Riemann–Liouville derivative, the Caputo derivative of a constant function is zero, just as in the case of integer-order differentiation.

2.3 Fractional Gradient Descent

We obtain the partial Caputo fractional derivative of a function $f : \mathbb{R}^n \to \mathbb{R}$ of order $\alpha \in (0, 1]$. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function with base point $a = (a_1, \ldots, a_n)$ such that $x_i > a_i$

for each i. Then the *Caputo partial fractional derivative* of f with respect to x_j can be defined as:

$$\frac{\partial_{a^+}^{\alpha_j} f(x)}{\partial x_j} = {}^{C}D_{a^+}^{\alpha_j} \left[f(x)\right]_j,$$

where ${}^{C}D_{a^+}^{\alpha_j}$ denotes the Caputo derivative of order α_j in the x_j–direction.

Similarly, for a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ with base point $a = (a_1, \dots, a_n)$, the Caputo fractional gradient of order $\alpha = (\alpha_1, \dots, \alpha_n)$ at $x = (x_1, \dots, x_n)$ can be defined as [12]:

$${}^{C}\nabla_{a^+}^{\alpha} f(x) = \begin{bmatrix} \dfrac{\partial_{a^+}^{\alpha_1} f(x)}{\partial x_1} \\ \vdots \\ \dfrac{\partial_{a^+}^{\alpha_n} f(x)}{\partial x_n} \end{bmatrix},$$

with each $\alpha_j > 0$ and $a_i < x_i$ for $i = 1, \dots, n$.

2.4 Fractional Gradient Descent Algorithm with the Caputo Derivative

The classical Taylor series can be extended to use Caputo fractional derivatives, allowing one to model dynamics with memory effects. In this context, the Caputo–Taylor expansion of a function f around t_0 is given by [12]:

$$f(t) \approx f(t_0) + \sum_{k=1}^{\infty} \frac{\left({}^{C}D_{t_0}^{\alpha} f\right)^{(k)}(t_0)}{\Gamma(\alpha k + 1)} (t - t_0)^{\alpha k}, \quad t_0 < t < t_0 + R, \text{ for some } R > 0, \tag{3}$$

where ${}^{C}D_{t_0}^{\alpha} f$ is the Caputo fractional derivative of f evaluated at t_0.

Let $x^{(k)} \in \mathbb{R}^n$ be the current iterate. Using the Caputo–Taylor expansion of order α about $x^{(k)}$ for the function f, we have:

$$f\big(x^{(k)} + h\big) \approx f\big(x^{(k)}\big) + \big({}^{C}\nabla^{\alpha} f\big)\big(x^{(k)}\big)^{\top} h,$$

where ${}^{C}\nabla^{\alpha} f(x^{(k)})$ is the Caputo fractional gradient of order α at $x^{(k)}$.

The direction that maximally decreases f locally, according to this approximation, is opposite to the Caputo fractional gradient, $-{}^{C}\nabla^{\alpha} f(x^{(k)})$. Thus, the iteration for the Caputo fractional gradient descent is:

$$x^{(k+1)} = x^{(k)} - \eta\, {}^{C}\nabla^{\alpha} f\big(x^{(k)}\big),$$

where $\eta > 0$ is the *learning rate.*

This algorithm mirrors the classical gradient descent, except that the integer–order gradient is replaced by the Caputo fractional gradient. To minimize $f(x)$, we move against the Caputo fractional gradient ${}^{C}\nabla^{\alpha} f(x)$.

The algorithm steps are the following:

1. **Input:**
    - Initial point $x^{(0)} \in \mathbb{R}^n$,

- Fractional orders $\alpha_j \in (0,1]$,
- Learning rate $\eta > 0$.

2. **For** $k = 0, 1, 2, \dots$ until convergence:
   - Compute $g_k = {}^{C}\nabla^{\alpha} f\big(x^{(k)}\big)$.
   - Update $x^{(k+1)} = x^{(k)} - \eta\, g_k$.

Algorithm 1: Caputo Fractional Gradient Descent

```
Data: $x^{(0)} \in \mathbb{R}^n,\ \varepsilon > 0,\ \alpha_j \in (0,1],\ 0 < \eta < 1$
Result: Approximate minimizer $x^*$
$k \leftarrow 0$
while $\|{}^{C}\nabla^{\alpha} f(x^{(k)})\| \geq \varepsilon$ do
    $x^{(k+1)} \leftarrow x^{(k)} \ - \ \eta\, {}^{C}\nabla^{\alpha}_{a^+} f\big(x^{(k)}\big)$
    $k \leftarrow k + 1$
end
```

3 Methodology

3.1 Network Architecture

We employ a fully-connected feed-forward neural network with

- Input: 5 features (S, K, τ, r, σ),
- Hidden layers: 3 layers of 32 neurons with ReLU activation,
- Output: linear neuron for price.

Weights $\mathbf{w}$ are randomly initialized.

Practical Considerations

- *Normalization:* Inputs and targets have been standardised between 0 and 1.

$$x = \big(S,\ K,\ \tau,\ r,\ \sigma\big) \ = \ (x_1, \dots, x_5) \in [0,1)$$

- *Activation Function:* We use the Rectified Linear Unit (ReLU) activation function. For the tests we have performed, ReLU gave us faster convergence than the Exponential Linear Unit (ELU) activation function.

3.2 Backpropagation Using the Fractional Gradient-Descent–Type Method

The backpropagation algorithm adjusts the weights of a neural network by minimizing a loss function through gradient computation. Incorporating fractional calculus into this process allows for more precise control over the training dynamics, offering advantages in convergence and regularization [13, 14].

The fractional gradient descent uses fractional derivatives to compute weight updates. A common definition is obtained via the Caputo fractional derivative:

$$ {}^{C}\nabla^{\alpha} f(x) \ = \ \frac{1}{\Gamma(1-\alpha)} \int_a^x \frac{\partial f(t)}{\partial t}\, (x-t)^{-\alpha}\, \mathrm{d}t, \quad \alpha \in (0,1], \tag{4}$$

where α controls the degree of fractional memory in the gradient calculation.

The weight w_{ij} is updated iteratively as:

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} \; - \; \eta \; {}^{C}\nabla^{\alpha} \frac{\partial L}{\partial w_{ij}}, \tag{5}$$

where η is the learning rate, L is the loss function, and ${}^{C}\nabla^{\alpha}$ denotes the Caputo fractional gradient.

In the following experiments, we use the gradient descent and the Caputo fractional gardient descent methods.

4 Numerical Experiments

4.1 Data Generation

For numerical simplicity, we will focus our experiments on the results obtained when pricing a European Call Option on a stock that pays no dividend. We keep all parameters fixed, except for the underlying price.

For the training data, we sample 10,000 observations

$$S \sim U(80, 120), \; K = 100, \; \tau = 0.25, \; r = 0.05, \; q = 0, \; \sigma = 0.3,$$

For the testing data, we sample 1,000 observations as follows

$$S \sim U(60, 140), \; K = 100, \; \tau = 0.25, \; r = 0.05, \; q = 0, \; \sigma = 0.3,$$

In both cases we compute the call option price C_{BS} via [8], and normalize all input parameters. Once we have the trained neural networks, we convert the results using the inverse standardisation process.

4.2 Training and Testing

We train three instances of our feed-forward neural network described in section 3.1, using (i) classical gradient descent and (ii) Caputo fractional gradient descent of order $\alpha = 0.95$ and (iii) Caputo fractional gradient descent of order $\alpha = 0.7$.

The training set is generated synthetically over a wide range of moneyness, times to maturity, interest rates and volatilities via the Black–Scholes formula. For both optimizers we fix the learning rate at $\eta = 0.17$, run for at most 5000 epochs, and employ an early-stopping rule that terminates training once the loss

$$\mathcal{L}(\mathbf{w}) < 10^{-4},$$

thereby ensuring that the training process converges.

After training completes, we evaluate each network on an out-of-sample test set of European call inputs. By feeding the test feature vectors x_{test} through the two learned models, we obtain predicted prices $\widehat{C}_{\text{classical}}$ and $\widehat{C}_{\text{frac}}$. We then compare these predictions to the true Black–Scholes prices to assess the relative accuracy and robustness of the classical gradient descent versus the fractional-order training regimes.

4.3 Results

In Table 1 we report the key training and generalization metrics for the classical gradient-descent (NN_GD) and two Caputo fractional-gradient-descent variants (NN_FGD with $\alpha = 0.95$ and $\alpha = 0.7$).

Metric	**NN GD**	**NN FGD** $\alpha = 0.95$	**NN FGD** $\alpha = 0.7$
Iterations	394	450	1445
Training_Time	5.49376	9.29676	31.0117
Training_Loss_MSE	9.95×10^{-5}	9.99×10^{-5}	9.97×10^{-5}
Test_RMSE	91.6264	77.7767	45.0879

Table 1: Comparison of results for classical and fractional-order gradient descent methods.

We highlight the following observations:

1. **Convergence behavior.** Classical Gradient Descent algorithm attains the stopping criterion in only 394 iterations (5.49 seconds), whereas the Caputo Fractional Gradient Descent with $\alpha = 0.95$ requires 450 iterations (9.30 seconds), and the Caputo Fractional Gradient Descent with $\alpha = 0.7$ requires 1445 iterations (31.01 seconds). There is an inverse relationship between the iteration count and time with the order α of the fractional derivative.

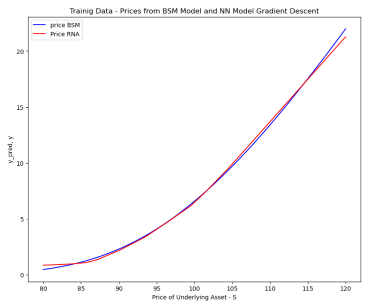
2. **Training-loss consistency.** All three methods drive the mean-squared training loss down to roughly 10^{-4}—specifically, 9.96×10^{-5}, 9.99×10^{-5}, and 9.98×10^{-5}. This near-equivalence in empirical risk indicates that each optimizer has obtained solutions of comparable quality in the training domain.

3. **Generalization improvement.** Despite similar training-loss performance, out-of-sample accuracy improves dramatically as the fractional order decreases: test root mean squared error falls from $RMSE = 91.63$ with the Classical Gradient Descent algorithm to $RMSE = 77.78$ with the Caputo Fractional Gradient Descent with $\alpha = 0.95$ and further to $RMSE = 45.09$ Caputo Fractional Gradient Descent with $\alpha = 0.7$. This pattern seems to indicate that the fractional-order optimisation process acts as an implicit regularizer, guiding the model toward minima that generalize more robustly.

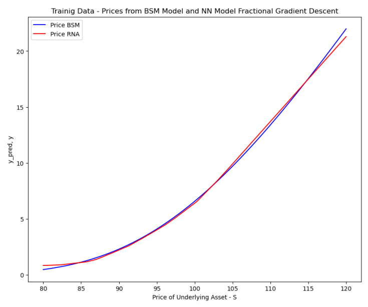
4. **Speed–accuracy trade-off.** The superior test accuracy of lower α values comes at the cost of more iterations and longer runtimes. In practice, one should treat α as a tunable hyperparameter. It would be expected thar some values of α may provide a favorable balance between computational efficiency and predictive robustness.

We present in Figure 1 a comparison of the options prices obtained with the Black–Scholes model and the prices obtained with the artificial neural networks, using the training dataset. While in Figure 2 we present a comparison of the options prices obtained with the Black–Scholes model and the prices obtained with the artificial neural networks, using the test dataset.

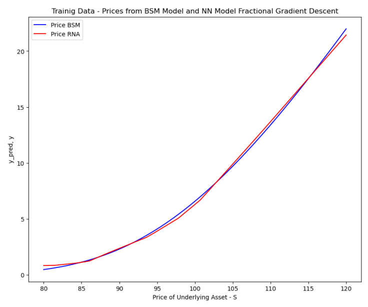
In these figures, (a) GD presents the Black–Scholes model (blue line) and the prices obtained with the artificial neural network trained using Classical Gradient Descent method (red line); (b) FGD $\alpha = 0.95$ presents the Black–Scholes model (blue line) and the prices obtained with the artificial neural network trained using Fractional Gradient Descent method with $\alpha = 0.95$ (red line); while (c) FGD $\alpha = 0.7$ presents the Black–Scholes model (blue line) and the prices obtained with the artificial neural network trained using Fractional Gradient Descent method with $\alpha = 0.7$ (red line).



(a) GD

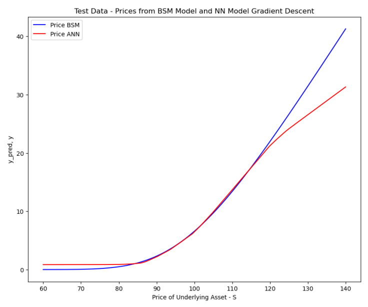


(b) FGD $\alpha = 0.95$

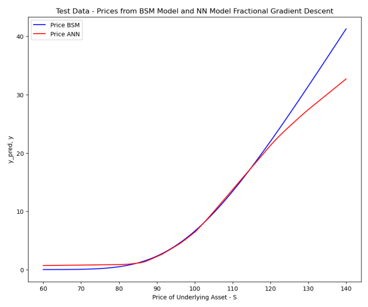


(c) FGD $\alpha = 0.7$

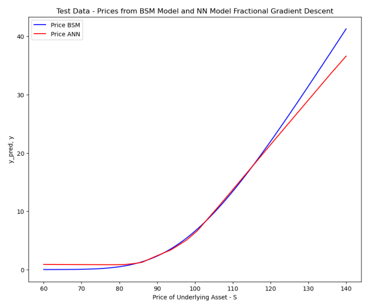
Figure 1: Comparison of BSM Model and NN Models using training data.



(a) GD



(b) FGD $\alpha = 0.95$



(c) FGD $\alpha = 0.7$

Figure 2: Comparison of BSM Model and NN Models using test data.

As can be observed in Figure 1, within the training dataset, there are no significant differences between the approximations obtained from the three neural networks we are evaluating. However, in Figure 2 there are observable differences between the results we obtained using out of sample data. Consistent with the results presented in Table 1, the artificial neural network that has been trained using the Fractional Gradient Descent method with $\alpha = 0.7$ can generalize better than the other two neural networks.

Considering these observations, the Fractional Gradient Descent algorithm seems to enhance out-of-sample performance via implicit memory-based regularization, while introducing a relative cost in training time.

5 Conclusions

In this work we presented a training paradigm for neural-network option pricing models based on *fractional gradient descent*, in which the classical weight update is replaced by a Caputo fractional gradient of order $\alpha \in (0, 1]$.

We presented the Caputo Fractional Gradient Descent algorithm and the Caputo derivative for the ReLU activation function. Then we implemented a fractional-order backpropagation algorithm that naturally incorporates a long-memory effect.

Numerical experiments on synthetic Black–Scholes data demonstrate that, although all methods achieve comparable training mean squared error as the loss function, decreasing the fractional order α yields dramatic improvements in out-of-sample accuracy: test root mean squared error (RMSE) is reduced by more than 50% when $\alpha = 0.7$ versus the classical case (i.e.,

$\alpha = 1$).

Our analysis indicates that fractional-order optimisation methods seem to act as an implicit regularizer, steering the optimizer toward minima that generalize more robustly to unseen market data. This benefit, however, comes with an increase in iterations and wall-clock time, highlighting a clear trade-off between convergence speed and generalization performance. In practical applications, one can treat α as a tunable hyperparameter or even schedule it adaptively during training to obtain a favorable balance.

Future research directions include:

- *Adaptive Fractional Orders:* Developing schemes that adjust α based on curvature or validation performance.
- *Rough-Volatility Calibration:* Extending the approach to calibrate rough-volatility models (e.g. rough Bergomi) using deep calibration frameworks.
- *Path-Dependent and Exotic Payoffs:* Evaluating fractional-order training on pricing and hedging of barrier, Asian, and other path-dependent options.
- *Theoretical Analysis:* Establishing convergence guarantees and generalization bounds for fractional descent in nonconvex settings.

Overall, fractional gradient descent enriches the toolkit for training financial neural networks. This optimisation method seems to embed memory-based regularization directly into the optimization dynamics, paving the way for both more accurate and more stable pricing engines in quantitative finance.

References

[1] J.M. Hutchinson, A.W. Lo, and T. Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *Journal of Finance*, 49(3):851–889, 1994.

[2] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. *MIT Press*, 2016.

[3] B. Horvath, A. Muguruza, and M. Tomás. Deep learning volatility: A deep neural network perspective on pricing and calibration in (rough) volatility models. *SSRN Electronic Journal*, pages 1–32, 2021.

[4] I. Podlubny. *Fractional Differential Equations*, volume 198 of *Mathematics in Science and Engineering.* Academic Press, 1999.

[5] E. C. Grigoletto and A. R. L. de Oliveira. Fractional order gradient descent algorithm. *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics*, pages 1–7, 2020.

[6] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[7] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[8] F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81:637–654, 1973.

[9] R.C. Merton. Theory of rational option pricing. *Bell Journal of Economics and Management Science*, 4:141–183, 1973.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015. arXiv:1412.6980.

[11] C. Bayer and B. Stemper. Deep calibration of rough stochastic volatility models. *arXiv preprint arXiv:1810.03399*, 2018.

[12] G. Candelario, A. Cordero, J. R. Torregrosa, and M. P. Vassileva. Generalized conformable fractional newton-type method for solving nonlinear systems. *Numerical Algorithms*, 93:1171–1208, 2023.

[13] M. R. Saleh and B. Ajarmah. Fractional gradient descent learning of backpropagation artificial neural networks with conformable fractional calculus. 2022.

[14] C. Bao, Y. Pu, and Y. Zhang. Fractional-order deep backpropagation neural network. *Computational Intelligence and Neuroscience*, 2018.