



Article

FAdamWav: A Fractional Wavelet Gradient Optimizer for Neural Networks

Oscar Herrera-Alcántara ^{1,*} , Salvador Arellano-Balderas ², Sandra Rodríguez-Mondragón ³,
José Alejandro Reyes-Ortíz ¹ and Jaime Navarro-Fuentes ²

¹ Departamento de Sistemas, Universidad Autónoma Metropolitana, Mexico City 02128, Mexico; jaro@azc.uam.mx

² Departamento de Ciencias Básicas, Universidad Autónoma Metropolitana, Mexico City 02128, Mexico; sab@azc.uam.mx (S.A.-B.); jnfu@azc.uam.mx (J.N.-F.)

³ Departamento de Evaluación del Diseño en el Tiempo, Universidad Autónoma Metropolitana, Mexico City 02128, Mexico; srm@azc.uam.mx

* Correspondence: oha@azc.uam.mx

Abstract

The optimizer is a critical element of neural networks because it computes their optimal parameters through a training process. The Adam optimizer is considered the state of the art in deep learning. However, a drawback is the cost of storing and computing their gradients. A useful tool for addressing this issue is the application of the wavelet transform, and other relevant tool is the fractional derivative, which can be used to create fractional gradient optimizers. This research combines the wavelet transform and fractional optimizers to propose FAdamWav, a fractional version of Adam that uses (i) a parametric discrete wavelet transform to theoretically save 50%, 75% or 87.5% of gradient's memory with one, two or three transformation levels, and (ii) a fractional gradient to optimize the neural network parameters. Experiments indicate that the saved memory is lower than the theoretical bounds, but memory is saved and fractional wavelet-based optimizers have competitive performance compared to their non-fractional and non-wavelet counterparts.

Keywords: machine learning; fractional derivative; gradient-based optimizers; wavelets

1. Introduction

In the context of artificial neural networks, an important component is the optimizer which can be used in the training process to obtain the optimal parameters of a neural network and extract intrinsic patterns of the input data. Due to the rapid development of Artificial Intelligence, several gradient-based optimizers have been proposed, starting with the fundamental Gradient Descent (GD) and evolving into more sophisticated versions with momentum and learning rate adaptation strategies.

Effectively, GD is the cornerstone for all the gradient-based optimizers given its fundamental principle of updating a parameter θ_t in the opposite direction to the gradient g_t . Mathematically, it can be expressed as follows:

$$\Delta\theta_t = -\eta g_t \quad (1)$$

where the learning rate η modulates the update for stepping towards getting a minimum [1]. The parameter update Formula (1) considers the entire training dataset and its know as the



Academic Editor: Ivanka Stamova

Received: 16 January 2026

Revised: 20 February 2026

Accepted: 24 February 2026

Published: 26 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Vanilla Gradient Descent optimizer (VGD), whereas the Stochastic Gradient Descent (SGD) updates the parameters for each i th training sample [2], and its corresponding formula is:

$$\Delta\theta_{t,i} = -\eta g_{t,i}. \quad (2)$$

Beyond SGD, Adagrad [3] calculates a diagonal matrix $G_{t,ii}$ which accumulates the sum of squares of the historical gradients to modify the adjustment of each parameter θ_i . Thus, the Adagrad optimizer equation is:

$$\Delta\theta_{t,i} = -\frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i} \quad (3)$$

where $\epsilon > 0$ is introduced to avoid the indetermination in case of a zero gradient.

The Adadelta optimizer [4] deals with a disadvantage of Adagrad concerning the loss of the “unit of measure”, given that η is adimensional in Equation (3). Therefore, the numerator η can be replaced with the RMS of $\Delta\theta$, up to $t - 1$, by considering that:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon} = \sqrt{\gamma E[\Delta\theta^2]_{t-1} + (1 - \gamma)\Delta\theta_t^2 + \epsilon} \quad (4)$$

whereas the denominator can be replaced with the latest RMS of g_t . Hence, the Adadelta formula is:

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t \quad (5)$$

where

$$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon}. \quad (6)$$

Adadelta considers an average window given $0 \leq \gamma \leq 1$ to ponderate the squares of current and previous one gradients according to the following formula:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma)g_t^2. \quad (7)$$

The sophisticated Adam optimizer [5] combines the average of past gradients described by:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t \quad (8)$$

and the past of squared gradients expressed as:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2 \quad (9)$$

appealing to the momentum concept. The formula for m_t contains a previous memory value m_{t-1} followed by a second term based on the gradient g_t .

Similarly, the velocity $v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$ uses a one-step memory update term and a squared gradient term. Since m_t and v_t are initialized to zero, to avoid a zero-bias tendency for the first momentum, a bias-corrected version is $\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$, and for the second momentum the bias-corrected version is $\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$. By considering this, the Adam update formula is:

$$\Delta\theta_{t,i} = -\frac{\eta}{\sqrt{\hat{v}} + \epsilon} \hat{m}_t \quad (10)$$

where a zero division is avoided with $\epsilon > 0$ in the denominator.

The Adam optimizer is considered the state of the art in deep learning due to its superior performance compared to others, once it computes the decay averages for the first and second moments of the gradients across many neural network layers.

However, a drawback of Adam is the high cost of storing and computing the historical moments.

Several techniques have been proposed for memory reduction in neural networks. For example, Low-Rank Adaptation (LoRA), which is a fine-tuning technique designed to adapt large pre-trained models to specific tasks by freezing a big model and training only small low-rank adapters to reduce gradient's memory requirements. LoRA is a weight-based approach to save memory [6].

Complementary to LoRA, there are gradient-based approaches, such as Low-Rank Gradient Projection (GaLore), which decomposes the update matrix, to reduce the optimizer state-memory usage by performing Singular Value Decomposition (SVD) on the gradients [7]. It focuses on exploiting the low-rank structure, within the gradient, to achieve memory reduction by projecting the gradient matrix $g_t \in R^{n \times m}$ into a low-dimensional subspace spanned by the columns of a projection matrix $P \in R^{m \times r}$ with $r \ll m$. Since gradients are required for parameter updates, the gradient is approximated as $(g_t P) P^{-1}$ by projecting the stored low-dimensional gradient $g_t P$ back to the original space [8]. Note that the success of LoRP is the projection to a lower dimension space, and to recover an approximation for the gradients.

In this regard, the wavelet transform is a useful tool which can be applied to reduce the memory required for the moments, based on the principle that energy can be concentrated in fewer coefficients while maintaining the performance of the neural network. A related work [9] presents the Gradient Wavelet Transform (GWT), where the wavelet transform is applied to gradients and significantly reduces the memory requirements of the optimizer at each training step.

The discrete wavelet transform can be defined as:

$$W_{m,n} = \frac{1}{\sqrt{a_0^m}} \int f(x) \psi\left(\frac{x - nb_0 a_0^m}{a_0^m}\right) dx \quad (11)$$

where $a_0 > 1$, $b_0 > 0$ and $m \times n \in \mathbb{Z}^2$ [10].

A well-known case is the dyadic wavelet transform where $a_0 = 2$ and $b_0 = 1$. In this case, the reconstruction for $f(x)$ can be expressed as:

$$f(x) = \sum_{m,n \in \mathbb{Z}} W_{m,n} 2^{-\frac{m}{2}} \psi(2^{-m}x - n) \quad (12)$$

where $2^{-\frac{m}{2}} \psi(2^{-m}x - n)$ constitutes an orthonormal basis for $L^2(\mathbb{R})$ and has good time-frequency localization properties. So, the wavelet coefficients $W_{m,n}$ concentrate the energy in low-frequency components through multiresolution analysis (MRA) [11]. In an MRA, $f(x)$ has successive coarser and coarser approximations averaging f over larger intervals, and at every step combining the difference between the approximation with resolution 2^{m-1} and the next coarser level, with resolution 2^m , as a combination of $W_{m,n}$ [10].

The successive resolution spaces are denoted by V_m where f is approximated as piecewise constant on the interval $[2^m n, 2^m(n+1)]$. These spaces have the following properties:

1. $\dots \subset V_2 \subset V_1 \subset V_0 \subset V_{-1} \subset V_{-2} \subset \dots$;
2. $\bigcap_{m \in \mathbb{Z}} V_m = \{0\}$, $\overline{\bigcup_{m \in \mathbb{Z}} V_m} = L^2(\mathbb{R})$;
3. $f \in V_j \iff f(2^m \cdot) \in V_0$;
4. $f \in V_0 \rightarrow f(\cdot - n) \in V_0$;
5. There exists $\phi \in V_0$ so that $\phi_{0,n}(x) = \phi(x - n)$, a scaling function, is an orthonormal basis for V_0 .

The MRA states that, given these five properties, there exists ψ such that $\psi(2^{-m}x - n)$ constitutes an orthonormal basis. In this way, the approximation $Proj_{V_{m-1}}$ at level $m - 1$ is given by:

$$Proj_{V_{m-1}}f = Proj_{V_m}f + \sum_{m \in \mathbb{Z}} \langle f(x), \psi(2^{-m}x - n) \rangle \psi(2^{-m}x - n). \quad (13)$$

For example, given the Haar basis, it is described by a scaling function ϕ :

$$\phi(x) = \begin{cases} 1, & \text{if } 0 \leq x < 1 \\ 0, & \text{otherwise} \end{cases} \quad (14)$$

together with a wavelet function ψ :

$$\psi(x) = \begin{cases} 1, & \text{if } 0 \leq x < \frac{1}{2} \\ -1, & \text{if } \frac{1}{2} \leq x < 1 \\ 0, & \text{otherwise.} \end{cases} \quad (15)$$

Then, the MRA is expressed as:

$$\phi(x) = \sum_n h_n \phi(2x - n), \quad (16)$$

$$\psi(x) = \sum_n g_n \phi(2x - n) \quad (17)$$

where $\phi_{-1,n}(x) = \sqrt{2}\phi(2x - n)$ represents an orthonormal basis for V_{-1} given the sequences $h_n = \sqrt{2} \langle \phi, \phi_{-1,n} \rangle$ and $g_n = (-1)^n h_{-n+1}$.

One of the most powerful results of MRA is the relationship with sub-band filtering, where a digital signal can be decomposed at different resolution levels by iteratively applying filter banks [11]. For the Haar basis, the filter bank is given by a pair of low-pass and high-pass filters. The low-pass filter corresponds to the sequence of coefficients $h_n = \{\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\}$, whereas the high-pass filter corresponds to the sequence $g_n = \{\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\}$. In this sense, the 2×2 Haar matrix is:

$$H_m = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (18)$$

which is conformed by two rows h_n and g_n , so that it represents a filter bank with perfect reconstruction given that each decomposition level is reversible via the inverse matrix $H_m^{-1} = H_m$.

The discrete wavelet transform for data D is obtained as $H_m \cdot D$, and the reconstruction is given by $H_m^{-1} \cdot H_m \cdot D$. The factor $\frac{1}{\sqrt{2}}$ for H_m preserves the norm of the transformed signal and provides perfect reconstruction.

Several perfect reconstruction filters have been studied as generalizations of Haar filters [12]. In particular, length-four filters can be obtained from a set of parametric formulas, such as:

$$\begin{aligned}h_0 &= \frac{1}{2\sqrt{2}} + \frac{1}{2} \cos \alpha \\h_1 &= \frac{1}{2\sqrt{2}} + \frac{1}{2} \sin \alpha \\h_2 &= \frac{1}{2\sqrt{2}} - \frac{1}{2} \cos \alpha \\h_3 &= \frac{1}{2\sqrt{2}} - \frac{1}{2} \sin \alpha\end{aligned}\tag{19}$$

which provide h_n coefficients that conform a low-pass filter. The corresponding high-pass filter is given by the set $g_0 = h_3, g_1 = -h_2, g_2 = h_1$ and $g_3 = -h_0$. Special cases for Equation (19) are the Haar filter and Dau4 filter, obtained with $\alpha = \frac{\pi}{4}$ and $\alpha = \frac{5\pi}{12}$, respectively. Other parameterizations have also been studied. For example, length-sixth filters have been studied in [13], length-eight and twelve in [14] and length-sixteen filters have been revised in [15].

Given these parametric filters, it is possible to implement a Parametric Discrete Wavelet Transform (PDWT) that can be applied to the Adam gradients to reduce the memory required for the update step.

For this research, a first goal is to apply the PDWT to the Adam gradients to concentrate the energy in a reduced number of coefficients. Thus, the updates for m_t and v_t are calculated in a reduced wavelet-space, where the gradients do not contain high frequencies. Then, the gradients are restored to the original space to update the network parameters for each training step. Moreover, a second goal is to apply fractional gradients, through the Caputo fractional derivative [16–18] to enhance the optimization performance [19–21].

Thus, in the present paper, a combination of parametric wavelet transform and fractional gradients are combined to propose FAdamWav, a fractional Adam optimizer where the wavelet transform is applied to reduce the memory of the momentums, and the fractional gradient is used to optimize the free parameters of neural networks.

With this in mind, the rest of the paper is organized as follows: In Section 2, the discrete wavelet transform is revised together with the Caputo fractional derivative and the Adam optimizer. This is the basis for introducing FAdam, an optimizer where a fractional gradient is applied to Adam to optimize the neural network parameters. Subsequently, the parametric wavelet transform is applied to historical momentums of FAdam to obtain FAdamWav, our fractional Adam wavelet-based optimizer.

In Section 3, several experiments are presented to support the main idea that a fractional wavelet optimizer:

- Introduces additional execution time due to the discrete wavelet transform, which has a linear computational complexity;
- Saves memory for the gradients by eliminating high-frequency sub-bands in the wavelet-space;
- Calculates efficiently the momentum m_t and velocity v_t in a low-dimensional wavelet-space;
- Calculates a non-perfect reconstruction for the gradients via the inverse parametric discrete wavelet transform. Then, the recovered approximations of gradients are used to update the neural network parameters;
- Allows to choose a parametric filter for the discrete wavelet transform to provide the best reconstruction for the gradients;

- Enhances the performance compared to its non-fractional wavelet counterparts by considering that fractional gradient optimizer includes the integer case ($\nu = 1.0$) as special case;
- Saves gradient's memory preserving competitive performance compared to the original case without wavelet transformation.

Finally, in Section 4, a discussion takes into account the theoretical concepts and the experimental results to see the advantages and disadvantages of applying our approach. Additionally, future work is commented.

2. Materials and Methods

As previously mentioned, two parts are considered to conform FAdamWav: (i) the parametric discrete wavelet transform that allows to reduce the memory of the Adam gradients for each update step and (ii) the application of the fractional gradient, which uses the Caputo definition of fractional derivative to update the Adam gradients.

2.1. Parametric Discrete Wavelet Transform

The one-dimensional parametric discrete wavelet transform (PDWT) is presented in Algorithm 1, which describes how to implement a perfect reconstruction transformation by applying a pair of parametric filters: a low-pass filter h and a high-pass filter g . The PDWT applies up to K decomposition levels (analysis) to data with $n = 2^K$ samples, which correspond to an MRA.

Figure 1 illustrates the PDWT with three decomposition levels. For $k = 1$, the sub-band L has low-pass wavelet coefficients (coarse information), whereas the sub-band H stores high-pass coefficients (detailed information). For $k = 2$, the second level is only applied to sub-band L , generating LL and LH sub-bands. For $k = 3$, the sub-band LL is transformed to obtain bands LLL and LLH , and so on, for further levels.

For each level, $k = 1, 2, \dots, K$, the number of low-pass wavelet coefficients L is reduced by half. The number of operations for convolution with the h and g filters is twice the length of data at the k th level, which has 2^{K-k+1} samples. Therefore, the complexity of the PDWT is $\sum_{k=1}^K 2^k$, yielding an efficient linear complexity of $O(n)$.

The high-frequency components are in sub-bands with at least one H . The low-frequency sub-band contains only L s. For two levels, the sub-band LL has low frequencies, and for three levels the LLL sub-band has low frequencies as well. It is worth mentioning that the low-frequency sub-band preserves much of the energy, defined as the sum of the squares of the samples. Additionally, for a single transformation level ($k = 1$), erasing the H -band saves 50% of the memory. For $k = 2$, erasing H and LH bands saves 75% of the memory. For $k = 3$, erasing the H , LH , and LLH bands saves 87.5% of the memory, and so on. Although the reconstruction is not perfect, selecting a parametric filter can concentrate more energy in the low-frequency sub-band for the given samples [22], promoting a good approximation to perfect reconstruction.

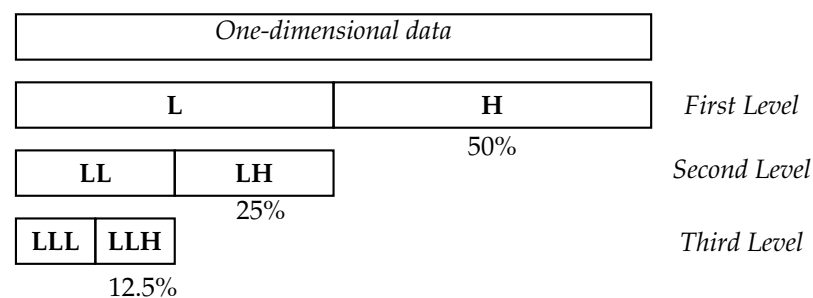


Figure 1. PDWT with three levels: Analysis (top-down), synthesis (bottom-up).

Algorithm 1 Parametric Discrete Wavelet Transform

Require: Signal array: *data*, Low-pass filter array of length FILTERSIZE: *h*, High-pass filter array of length FILTERSIZE: *g*, Number of wavelet decomposition levels: LEVELS

```

1: procedure PDWT(data, data.length : n, FILTERSIZE, h, g, LEVELS)
2:   for level = 0 to LEVELS do
3:     DWTLEVEL(data, data.length  $\gg$  level, FILTERSIZE, h, g)
4:   end for
5: end procedure

6: procedure DWTLEVEL(data, data.length:n, FILTERSIZE, h, g)
7:   if n  $\geq$  FILTERSIZE then
8:     i  $\leftarrow$  0, j  $\leftarrow$  0
9:     tmp  $\leftarrow$  new array of size n  $\triangleright$  Temporal array for wavelet coefficients
10:    for j = 0 to n − 2 step 2 do
11:      tmp[i]  $\leftarrow$  0
12:      tmp[i + n/2]  $\leftarrow$  0
13:      for z = 0 to FILTERSIZE − 1 do
14:        tmp[i] + = data[(j + z) mod n] · h[z]  $\triangleright$  Low frequency coefficients
15:        tmp[i + n/2] + = data[(j + z) mod n] · g[z]  $\triangleright$  High frequency coefficients
16:      end for
17:      i  $\leftarrow$  i + 1
18:    end for
19:    data  $\leftarrow$  tmp  $\triangleright$  Updated wavelet coefficients
20:  end if
21: end procedure

```

For reconstruction (synthesis), the same number of levels must be applied as for PDWT. These levels correspond to the Parametric Inverse Discrete Wavelet Transform (PIDWT), which is described in Algorithm 2.

Algorithm 2 Parametric Inverse Discrete Wavelet Transform

Require: Signal array: *data*, Low-pass filter array of length FILTERSIZE: *Ih*, High-pass filter array of length FILTERSIZE: *Ig*, Number of wavelet decomposition levels: LEVELS

```

1: Ih, Ig  $\leftarrow$  INVERSEFILTERS(FILTERSIZE, h, g)
2: procedure PIDWT(data, data.length : n, FILTERSIZE, Ih, Ig, LEVELS)
3:   for level  $\leftarrow$  LEVELS downto 0 do
4:     INVTWDLEVEL(data, data.length  $\gg$  level, FILTERSIZE, h, g)
5:   end for
6: end procedure

7: procedure INVTWDLEVEL(data, data.length, FILTERSIZE, Ih, Ig)
8:   if n  $\geq$  filtersize then
9:     ndiv2  $\leftarrow$  n/2
10:    tmp  $\leftarrow$  new array of size n
11:    j  $\leftarrow$  FILTERSIZE − 2
12:    for i  $\leftarrow$  0 to ndiv2 − 1 do
13:      tmp[j mod n]  $\leftarrow$  0
14:      for w  $\leftarrow$  0, q  $\leftarrow$  0 to filtersize/2 − 1 do
15:        ii  $\leftarrow$  (i + w/2) mod ndiv2
16:        tmp[j mod n] + = data[ii] · Ih[w]
17:        tmp[j mod n] + = data[ii + ndiv2] · Ih[w + 1]
18:        w  $\leftarrow$  w + 2
19:      end for
20:      j  $\leftarrow$  j + 1
21:      tmp[j mod n]  $\leftarrow$  0
22:      for w  $\leftarrow$  0 to FILTERSIZE − 1 step 2 do
23:        tmp[j mod n] + = data[(i + w/2) mod ndiv2] · Ig[w]
24:        tmp[j mod n] + = data[(i + w/2) mod ndiv2 + ndiv2] · Ig[w + 1]
25:      end for
26:      j  $\leftarrow$  j + 1
27:    end for
28:    data  $\leftarrow$  tmp
29:  end if
30: end procedure

```

For each transformation level the number of wavelet coefficients is duplicated up to the original via the PIDWT using two reconstruction filters *Ih* and *Ig* obtained from *h* and *g*, according to Algorithm 3.

Algorithm 3 Reconstruction filters

```

1: function INVERSEFILTERS(FILTERSIZE,  $h, g$ )
2:   for  $i \leftarrow 0$  to FILTERSIZE − 1 do
3:      $base \leftarrow \begin{cases} h & \text{if } i \text{ is even} \\ g & \text{if } i \text{ is odd} \end{cases}$ 
4:      $Ih[i] \leftarrow base[2 \cdot \lfloor (FILTERSIZE - 1 - i)/2 \rfloor]$ 
5:      $Ig[i] \leftarrow base[2 \cdot \lfloor (FILTERSIZE - 1 - i)/2 \rfloor + 1]$ 
6:   end for
7:   return  $Ih[i], Ig[i]$ 
8: end function

```

The shortest filter has a length of two and corresponds to the Haar filter. Parametric length-four filters are shown in Equation (19).

Length-sixth filters are described in Equation (20), depending on two parameters α and β as follows:

$$\begin{aligned}
 p &= 0.5 * \sqrt{1 + \sin(\alpha + \frac{\pi}{4})} \\
 h[0] &= \sqrt{2} \cdot \left(\frac{1}{8} + \frac{1}{4\sqrt{2}} \cos(\alpha) + \frac{p}{2} \cos(\beta) \right) \\
 h[1] &= \sqrt{2} \cdot \left(\frac{1}{8} + \frac{1}{4\sqrt{2}} \sin(\alpha) + \frac{p}{2} \sin(\beta) \right) \\
 h[2] &= \sqrt{2} \cdot \left(\frac{1}{4} - \frac{1}{2\sqrt{2}} \cos(\alpha) \right) \\
 h[3] &= \sqrt{2} \cdot \left(\frac{1}{4} - \frac{1}{2\sqrt{2}} \sin(\alpha) \right) \\
 h[4] &= \sqrt{2} \cdot \left(\frac{1}{8} + \frac{1}{4\sqrt{2}} \cos(\alpha) - \frac{p}{2} \cos(\beta) \right) \\
 h[5] &= \sqrt{2} \cdot \left(\frac{1}{8} + \frac{1}{4\sqrt{2}} \sin(\alpha) - \frac{p}{2} \sin(\beta) \right).
 \end{aligned} \tag{20}$$

It is worth mentioning that special cases for Equation (19) are Haar and Dau4, as is shown in Table 1, with the corresponding parameter α [23].

Table 1. Special cases for length-four parametric filters.

Filter	α
Haar	0.7853981633974483
Dau4	1.3089969389957472

Additionally, there are special cases of length-sixth parametric filters of Equation (20) for Haar, Dau4, Dau6, and Coiflets, which are listed in Table 2 with the corresponding α and β parameters [23].

Table 2. Special cases for length-sixth parametric filters.

Filter	α	β
Haar	0.7853981633974483	0.7853981633974483
Dau4	1.3089969389957472	1.0471975511965976
Dau6	1.7850806990394015	1.0742468359786252
Coiflets	5.7504673989251085	3.6052402624389956

By selecting one of these parameterizations for the low-pass h and high-pass g filters, the parametric discrete wavelet transform can be applied to the gradients of the Adam optimizer.

2.2. Fractional Derivatives and the Backpropagation Update Formula for MLP

There is no single definition of fractional derivatives. Three notable examples are the Grünwald–Letnikov, the Riemann–Liouville, and the Caputo fractional derivatives [16,19,24] which are defined and described briefly below.

Definition 1. The Grünwald–Letnikov derivative with order $\nu > 0$ of a given function $f(x)$, $x \in (a, b)$ is defined as:

$${}_a^{\text{GL}}D_x^\nu f(x) = \lim_{\substack{h \rightarrow 0 \\ Nh = x-a}} h^{-\nu} \sum_{j=0}^N (-1)^j \binom{\nu}{j} f(x - jh). \quad (21)$$

Definition 2. The Riemann–Liouville derivative with order $\nu > 0$ and $n = [\nu + 1]$ of a given function $f(x)$, $x \in (a, b)$ is defined as:

$${}_a^{\text{RL}}D_x^\nu f(x) = \frac{1}{\Gamma(n - \nu)} \frac{d^n}{dx^n} \int_a^x (x - y)^{n-\nu-1} f(y) dy. \quad (22)$$

Definition 3. The Caputo fractional derivative, for $a, x \in \mathbb{R}$, $\nu > 0$ and $n = [\nu + 1]$, is defined as:

$${}_a^{\text{C}}D_x^\nu f(x) = \frac{1}{\Gamma(n - \nu)} \int_a^x (x - y)^{n-\nu-1} f^{(n)}(y) dy. \quad (23)$$

Although these fractional derivatives are certainly related, they are not equivalent [25]. For example, the Riemann–Liouville definition is related to the Caputo definition as follows:

$${}_a^{\text{RL}}D_x^\nu f(x) = {}_a^{\text{C}}D_x^\nu f(x) + \sum_{k=0}^{n-1} \frac{f^{(k)}(a)(x-a)^{k-\nu}}{\Gamma(k+1-\nu)}, \quad (24)$$

and, for a suitably smooth function $f(x)$, i.e., $f \in C^m[a, b]$, then

$${}_a^{\text{RL}}D_x^\nu f(x) = {}_a^{\text{GL}}D_x^\nu f(x). \quad (25)$$

From the definition of the Riemann–Liouville derivative, it follows that if $f(x) = k$ is a non-zero constant, then:

$${}_0^{\text{RL}}D_x^\nu f(x) = \frac{k}{\Gamma(1-\nu)} x^{-\nu}, \quad (26)$$

which is not zero and is different from the integer case.

In this paper and other works, the Caputo definition is preferable to the Grünwald–Letnikov and Riemann–Liouville definitions because the Caputo fractional derivative of Equation (23) is zero for $f(x) = k$, with $k \in \mathbb{R}$. This matches the integer derivative version [18].

Based on the previous discussion, we use the Caputo derivative definition to obtain a fractional expression for the gradients of the Adam algorithm. As we pointed out, the Caputo derivative of a constant is zero and for $f(x) = x^p$ [18,26]:

$${}_0^{\text{C}}D_x^\nu x^p = \frac{\Gamma(p+1)x^{p-\nu}}{\Gamma(p-\nu+1)}. \quad (27)$$

In the case of $p = 1$, it yields to:

$${}_0^{\text{C}}D_x^\nu x = \frac{x^{1-\nu}}{\Gamma(2-\nu)}. \quad (28)$$

Given this fractional derivative and the backpropagation formula which is used to update the parameters of the Multilayer Perceptron (MLP) [1]:

$$\Delta w_{kj}^l = -\eta \frac{\partial E_i}{\partial w_{kj}^l} = -\eta D_{w_{kj}^l} E_i \quad (29)$$

our fractional version to update the gradients for FAdamWav is:

$$D_{w_{kj}^l}^\nu E_i = \frac{\partial E_i}{\partial w_{kj}^l} \cdot D_{w_{kj}^l}^\nu w_{kj}^l = \delta_k^l \cdot a_j^{l-1} \cdot \frac{(w_{kj}^l)^{1-\nu}}{\Gamma(2-\nu)}. \quad (30)$$

See Appendix A for details. Now, it is worth making a comment in this regard. In the context of the calculus of paradifferential operators in Sobolev spaces, and in this case for the Fourier multiplier operator D^ν , defined by:

$$(\widehat{D^\nu f})(\xi) = |\xi|^\nu \hat{f}(\xi), \quad (31)$$

the fractional chain rule [27,28] should be:

$$D^\nu F(u) = F'(u) D^\nu u + E, \quad (32)$$

where E is an error as per the result of Christ and Weinstein [29]. Thus, by assuming $E = 0$, it is an approximation, useful for our purposes. Suppose that $F \in C^1(\mathbb{C})$, $1 < p, q < \infty$, $\nu \in (0, 1)$, $r^{-1} = p^{-1} + q^{-1}$. If $u \in L^\infty(\mathbb{R})$, $D^\nu u \in L^q$, and $F'(u) \in L^p$, then $(D^\nu(F(u))) \in L^r$, and $\|D^\nu F(u)\|_r \leq C \|F'(u)\|_p \|D^\nu u\|_q$.

It is important to note that Equation (30) is the fractional version of Equation (A5), only different by the factor $\frac{(w_{kj}^l)^{1-\nu}}{\Gamma(2-\nu)}$ which is obtained when Equation (28) is applied to w_{kj}^l . Indeed, if $\nu = 1$, Equation (30) becomes the classical integer case. Then, Equation (30) represents a gradient-descent generalization with fractional order ν .

In practice, it is necessary to avoid two conditions:

- When the synaptic weights are zero, that yields to the indetermination of $\frac{(w_{kj}^l)^{1-\nu}}{\Gamma(2-\nu)}$ for $1 - \nu < 0$.
- When $1 - \nu$ is rational, we let $1 - \nu = \frac{r}{s}$ and s is even (for example $r = 1$ and $s = 2$). Then, if $w_{kj}^l < 0$, complex values are generated.

These situations have been previously studied in [20] and a solution consists of replacing w_{kj}^l by $|w_{kj}^l| + \epsilon$, for $\epsilon > 0$. In this way, the fractional gradient factor f_w^ν is defined as:

$$f_w^\nu := \frac{(|w_{kj}^l| + \epsilon)^{1-\nu}}{\Gamma(2-\nu)} \quad (33)$$

and the limit exists, and is equal to 1, for f_w^ν as $\nu \rightarrow 1$.

With this consideration, Equation (30) becomes:

$$D_{w_{kj}^l}^\nu E_i = \delta_k^l \cdot a_j^{l-1} \cdot f_w^\nu = \delta_k^l \cdot a_j^{l-1} \cdot \frac{(|w_{kj}^l| + \epsilon)^{1-\nu}}{\Gamma(2-\nu)} \quad (34)$$

which generalizes the integer gradient-descent update rule, and it allows to compute the fractional gradient for the Adam optimizer by including the factor f_w^ν .

The following Algorithm 4 presents the FAdamWav steps. In this case, the gradient g_t is a bidimensional $m \times n$ matrix. In Step 1, the fractional gradient is obtained by applying Equation (34). To warrant a two-power for the n -dimension, zero padding is

applied to obtain $n = 2^K$ samples. Then, the PDWT is applied to m -tensors and therefore the computational complexity to calculate the PDWT is $O(m \times n)$.

In this paper, it is meaningless to assume that a 2D wavelet transform on g_t could be better to exploit some kind of "curves" inside g_t . Instead, we just consider m transformations on n -dimensional tensors. However, we mention in brief a 2D approach, where a PDWT could be applied in two separated dimensions through tensor products of parametric filters or from non-tensor product of wavelet filters [30]. Indeed, it is possible to consider curvelets [31] and shearlets [32]. Curvelets are efficient transforms designed to work on images with edges along smooth curves, and they extend wavelets by incorporating directional selectivity and anisotropic scaling. The shearlet transform was designed to capture anisotropic features (like edges) in multidimensional data. We remark that these approaches are not considered in this paper, but it could be explored in future work.

Continuing with the FAdamWav algorithm (Algorithm 4), Step 5 shows L_t which represents the low-frequency band (only "L's") along T iterations and the H_t sub-bands with high-frequency components, for k transformation levels on g_t . After applying the PDWT, the H_t sub-bands are erased to save memory. Then, the m_t and v_t matrices are initialized to zero, and they are updated with L_t in Steps 10 and 11, by considering Equations (8) and (9), respectively. L_t has dimension of $m \times 2^{K-k+1}$ and m_t and v_t also save space since they have the same dimension. L_t reduces the low-frequency coefficients by half for each level. For example, when $k = 1$, the memory of the fractional gradients is reduced by 50%, g_t has a dimension $m \times \frac{m}{2}$, and H_t is set to zero in Step 6. So, in Step 12, L_t is calculated using Equation (10) according to Adam's update formula. One difference from related work [9] is that our algorithm fully erases H_t and is not scaled by the formula $\frac{H_t}{\sqrt{v_t} + \epsilon}$ which scales the high-frequency sub-bands with the v_t matrix to deal with noisy gradients. In our research, we assume the existence of an optimal parametric filter to compress the gradients, allowing for non-perfect reconstruction of g_t to update W_t parameters, as shown in Step 15. The main idea is that it is possible to achieve a competitive accuracy relative to the original g_t saving memory.

Algorithm 4 FAdamWav: Fractional Adam with Wavelet Transform

Require: Weight matrix W , learning rate η , batch size m . Adam decay rates β_1, β_2 . Iteration T . $\epsilon > 0$. Scale factor α . Lowpass wavelet filter h , highpass wavelet filter g , Levels k .

```

1: Initialize  $t \leftarrow 0$ 
2: repeat
3:    $g_t \leftarrow g_t \cdot f_w^\nu$  ▷ Fractional gradient. See Equation (37)
4:    $g_t \leftarrow \text{pad\_to\_pow2}(g_t)$  ▷ Save original length.  $\text{Length}(g_t) = 2^K$ 
5:    $[L_t, H_t] \leftarrow \text{PDWT}(g_t, \text{length}(g_t), \text{filtersize}, h, g, k)$  ▷ Gradient PDWT- Analysis
6:    $H_t \leftarrow 0$  ▷ No longer used, memory reduction
7:   if  $t = 0$  then
8:     Initialize  $m_{-1}, v_{-1} \leftarrow 0$ 
9:   end if
10:   $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) L_t$ 
11:   $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) L_t^2$ 
12:   $[L_t, H_t] \leftarrow \left[ \frac{m_t}{\sqrt{v_t} + \epsilon}, \frac{H_t}{\sqrt{v_t} + \epsilon} \right]$  ▷  $H_t$  was set to zero. Update rule, see Equation (10)
13:   $g_t \leftarrow \text{PIDWT}([L_t, H_t], \text{length}(L_t, H_t), \text{filtersize}, h, g, k)$  ▷ PIDWT - Synthesis. Back to the original length
14:   $\eta_t \leftarrow \eta \cdot \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t}$  ▷ Bias correction
15:   $W_t \leftarrow W_{t-1} - \eta_t \cdot g_t$  ▷ Update weights. Original dimensions for fractional gradients.
16:   $t \leftarrow t + 1$ 
17: until  $t = T$ 
18: return  $W_t$ 

```

Based on the previous concepts and Algorithm 4, the original PyTorch code for the Adam class [33] can be modified to obtain our fractional FAdamWav class, which is described in Listing 1. Specifically, by inserting Lines 8 to 20 as follows:

Listing 1. FAdamWav Class: PyTorch Source Code

```

1  class FAdamWav(Optimizer):
2      def step(self, closure: Callable = None):
3          ...
4          for group in self.param_groups:
5              for p in group["params"]:
6                  ...
7                  grad = p.grad
8                  grad *= torch.pow(abs(grad)+ epsilon, 1-nu )
9                  grad /= torch.exp(torch.lgamma(torch.tensor( 2.0-nu ) ))
10                 gr, original_lengths = pad_batch_to_pow2(grad)
11                 pwt = ParametricWaveletTransform ()
12                 gr = pwt.PDWT(gr, len(gr))
13                 gr= gr [..., :gr.shape[-1] // 2]
14                 grad = gr
15                 ...
16                 gr=norm_grad
17                 gr = duplicate_with_zeros(gr)
18                 gr = pwt.PIDWT(gr, len(gr))
19                 gr= restore_original_lengths(gr, original_lengths)
20                 grad = gr.squeeze(0)
21                 norm_grad = grad

```

In this case, the fractional factor depending of the ν -order is applied to $grad$ (see Lines 8 and 9), and consequently it becomes a fractional gradient. (The source code for FAdamWav, PDWT, and PIDWT is available at <http://ia.azc.uam.mx/sourcecode>) (accessed on 23 February 2026).

In the same code, Lines 10 to 20 apply the parametric discrete wavelet transform. Line 12 applies the PDWT. Note that Line 13 considers low-pass wavelet coefficients exclusively, under the principle that these coefficients should concentrate much of the gradient energy (high-pass wavelet coefficients are neglected) which reduces the memory required for the gradients as previously explained. In Line 17, after using the gradients to update the synaptic weights, the high-pass wavelet coefficients H_t are set to zero, and the PIDWT is applied in Line 18. Of course, this produces a non-perfect reconstruction, but it is sufficient to update the gradients of Adam since m_t and v_t are updated efficiently in the transformed space. The accuracy of the neural network should be lower than using the original gradient g_t . However, the main idea is to maintain competitiveness while saving memory. This idea is supported by experimental results.

3. Results

This section describes four experiments that support the conclusions.

- Experiment 1. (FAdam). It explores the accuracy of a neural network with the following architecture:

```

Linear(784, 64)
ReLU()
Linear(64, 10)

```

and fractional gradients $\nu \in (0, 2.0)$ on the MNIST dataset [34], without applying the parametric discrete wavelet transform. The neural network architecture is enough to match the input size of MNIST images (28×28 pixels) and the output with 10 classes. The MNIST dataset has 60,000 images, divided into a training set of 50,000 and a test set of 10,000.

- Experiment 2. (FAdamWav: Length-four filters). Given the same neural network and dataset as in Experiment 1, the second experiment applies the parametric wavelet transform to reduce the FAdam gradient memory by half by considering only the low-pass coefficients and a single transformation level. Several length-four parametric

filters are tested varying the parameter $\alpha = j \cdot \frac{\pi}{12}$ for $j = \{1, 2, 3, 4, 5, 6\}$. In this case, the best ν -value from Experiment 1 is chosen (see below $\nu = 0.2$).

- Experiment 3. (FAdamWav: Length-sixth filters). This experiment is similar to Experiment 2, but it considers length-sixth filters of Equation (20) depending on α and β . Each parameter independently takes on values equal to $j \cdot \frac{\pi}{12}$, where $j = \{1, 2, 3, 4, 5, 6\}$.
- Experiment 4. (FAdam, FAdamWav_k1, FAdamWav_k2, FAdamWav_k3). This experiment measures and compares the amount of GPU memory required for Adam and FAdamWav applying $k = 1, 2, 3$ wavelet transformation levels. The execution time is also compared.

3.1. Experiment 1

Table 3 shows the accuracy of the neural network trained using FAdam and different fractional gradients $\nu \in (0, 2.0)$, with step 0.2 along 20 epochs. The maximum accuracy of 98.36% is reached with $\nu = 0.2$, which is highlighted in bold.

Table 3. Neural Network accuracy for different ν -values with the FAdam optimizer.

$\nu = 0.2$	$\nu = 0.4$	$\nu = 0.6$	$\nu = 0.8$	$\nu = 1.0$	$\nu = 1.2$	$\nu = 1.4$	$\nu = 1.6$	$\nu = 1.8$
96.25	96.46	96.53	96.04	96.44	96.34	96.89	95.84	96.03
97.12	97.11	97.37	97.54	97.09	97.24	96.51	97.04	96.93
97.49	97.28	97.30	97.54	97.51	97.47	97.15	97.41	96.99
97.60	97.63	97.63	97.67	97.66	97.55	97.43	97.33	97.10
97.63	97.77	97.75	97.88	97.37	98.18	97.98	97.86	97.23
97.33	97.69	97.63	98.08	97.63	97.88	97.69	97.63	97.29
97.68	97.64	97.78	97.48	97.66	97.95	97.57	97.61	97.39
97.76	97.69	97.80	97.59	97.59	97.72	97.84	97.60	97.39
97.88	97.26	97.87	98.01	98.05	97.94	97.83	97.88	97.71
97.84	97.36	98.00	98.10	97.88	97.84	98.09	98.05	97.45
97.65	97.69	97.63	97.27	97.69	97.98	97.89	97.94	97.23
97.86	98.07	98.00	97.90	97.48	98.04	97.91	97.86	97.59
98.00	97.75	97.88	97.92	97.84	97.98	97.65	97.74	97.76
98.07	97.85	97.96	98.02	97.73	97.96	97.67	97.81	97.63
97.68	97.55	97.91	98.18	97.99	97.72	97.89	97.63	97.41
97.99	97.91	97.95	97.67	97.73	97.88	97.83	97.63	97.59
97.91	98.01	97.82	97.55	97.88	97.94	97.86	97.85	97.84
97.93	98.12	97.90	98.04	97.86	97.55	97.76	97.98	97.73
98.36	98.12	97.94	98.23	98.04	98.05	98.06	97.83	97.75
97.80	97.94	97.74	97.90	98.00	97.89	98.09	97.94	97.58

Figure 2 shows boxplots of the accuracies of Experiment 1 (with a step size of 0.1 for ν), showing that the best performance is achieved with $\nu = 0.2$. Additionally, the plot shows that the integer case $\nu = 1.0$ has lower performance than several fractional cases, including the case $\nu = 0.2$.

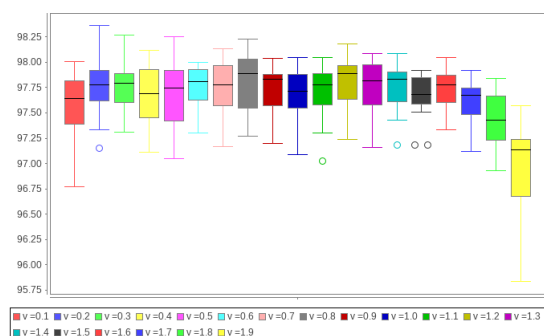


Figure 2. FAdamWav accuracies for $\nu \in [0.1, 1.9]$ without PDWT.

To complete Experiment 1, Figure 3 compares the performance of the integer case $\nu = 1.0$ against the case $\nu = 0.2$. It is notorious that the fractional gradient reaches a higher accuracy than the integer gradient.

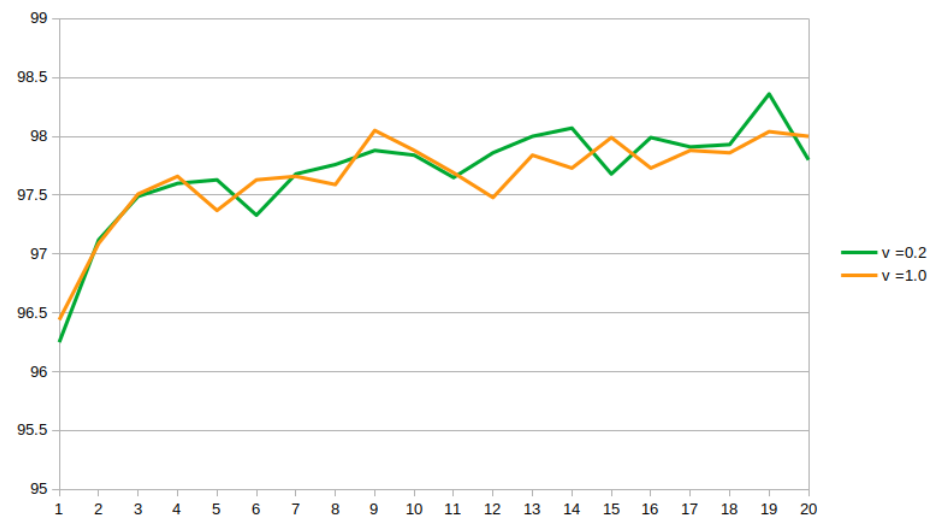


Figure 3. FAdam's accuracy for $\nu = 0.2$ (green) vs. $\nu = 1.0$ (orange).

3.2. Experiment 2

The second experiment applies a single level of the parametric wavelet transform to reduce the FAdam gradient memory by half and by considering only the low-pass wavelet coefficients. Then, the gradients are updated, the high-pass coefficients are set to zero, and the parametric inverse wavelet transform is calculated to recover the original size of the Adam gradients.

Several length-four parametric filters were tested by varying the parameter $\alpha = k \cdot \frac{\pi}{12}$ for $k = \{1, 2, 3, \dots, 12\}$. The ν -value was fixed to $\nu = 0.2$. The corresponding accuracies are shown in columns of Table 4, where the highest accuracy 98.16% was achieved with $\alpha = \frac{\pi}{6}$ and is highlighted in bold.

Table 4. Neural Network accuracy for $\nu = 0.2$ with FAdamWav optimizer: Several α values.

$\frac{\pi}{12}$	$\frac{\pi}{6}$	$\frac{\pi}{4}$	$\frac{\pi}{3}$	$\frac{5\pi}{12}$	$\frac{\pi}{2}$	$\frac{7\pi}{12}$	$\frac{2\pi}{3}$	$\frac{3\pi}{4}$	$\frac{5\pi}{6}$	$\frac{11\pi}{12}$	π
94.90	95.33	95.78	95.61	94.95	95.33	95.67	95.14	95.41	95.49	95.36	95.64
96.47	96.62	96.62	96.65	96.80	96.63	96.80	96.85	96.75	96.57	96.57	96.89
96.91	97.28	97.33	97.06	97.09	97.15	97.36	97.13	97.09	97.22	97.20	97.35
97.47	97.55	97.60	97.25	97.10	97.31	97.55	97.24	97.23	97.59	97.42	97.22
97.14	97.59	97.49	97.78	97.10	97.69	97.23	97.47	97.38	97.23	97.38	97.75
97.50	97.74	97.42	97.86	97.82	97.77	97.85	97.90	97.37	97.68	97.21	97.58
97.44	97.12	97.45	97.88	97.62	97.96	97.93	97.45	97.63	97.63	97.85	97.75
97.76	97.48	97.48	97.68	97.08	97.53	97.81	97.61	97.71	97.47	97.42	97.79
97.87	97.88	97.60	97.59	97.75	97.75	97.21	97.74	97.81	97.49	97.76	97.26
97.80	97.84	97.73	97.81	97.82	97.79	97.90	97.52	97.56	97.85	97.74	97.97
97.72	97.80	97.88	97.99	97.48	97.73	97.57	97.40	97.49	97.81	97.67	97.69
97.81	97.64	97.74	97.54	97.48	97.75	97.75	97.91	97.72	97.87	97.78	97.61
97.94	97.84	97.66	98.13	97.85	97.84	97.84	97.92	97.97	97.96	97.63	97.78
97.81	97.95	97.61	97.68	97.87	97.60	97.50	97.86	97.74	97.90	97.72	97.69
98.01	97.78	97.06	97.96	97.54	98.06	97.85	97.88	97.83	97.69	97.82	97.86
97.88	97.72	98.00	97.92	97.27	98.07	97.51	97.97	97.77	97.80	97.72	97.83
97.79	97.66	98.15	98.03	97.82	98.05	98.03	97.98	97.92	97.85	97.99	97.94
97.89	97.85	97.77	97.95	98.00	97.67	97.95	97.73	97.90	97.92	97.51	97.77
97.84	98.16	97.97	97.89	97.86	98.03	97.96	97.78	97.70	98.04	98.07	98.04
97.85	97.99	97.88	97.24	97.98	97.84	97.95	97.91	97.86	98.01	97.90	98.01

Figure 4 shows the boxplots of Table 4, and effectively it illustrates that the best performance occurs at $\alpha = \frac{\pi}{6}$.

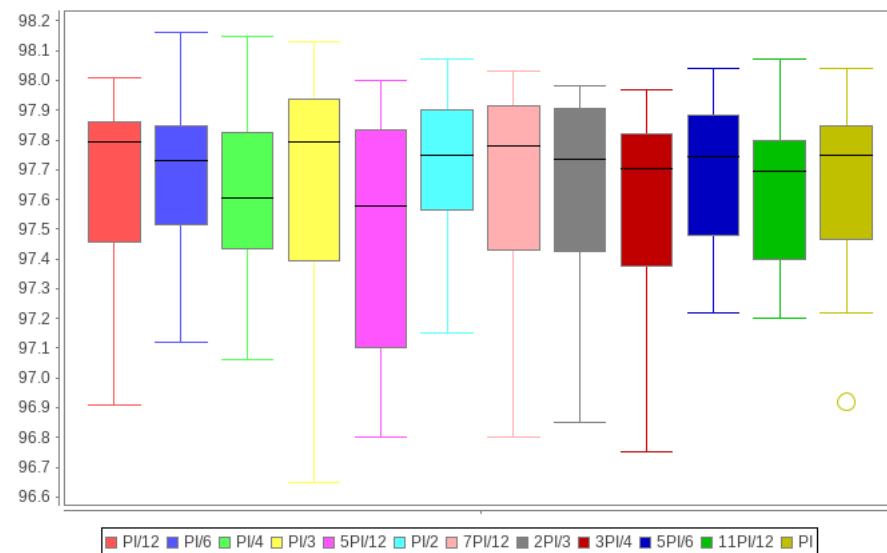


Figure 4. FAdamWav with length four filters, $\alpha = k \cdot \frac{\pi}{12}$ with $k \in \{1, 2, 3, \dots, 12\}$.

Figure 5 compares the accuracies of Fractional Adam (FAdam) with $\nu = 0.2$ and Fractional Adam with parametric wavelet transform (FAdamWav) with $\nu = 0.2$ and a parametric filter with $\alpha = \frac{\pi}{6}$. Although FAdam has a slightly better performance than FAdamWav, performance is still competitive but the memory required by the gradients is reduced by half.

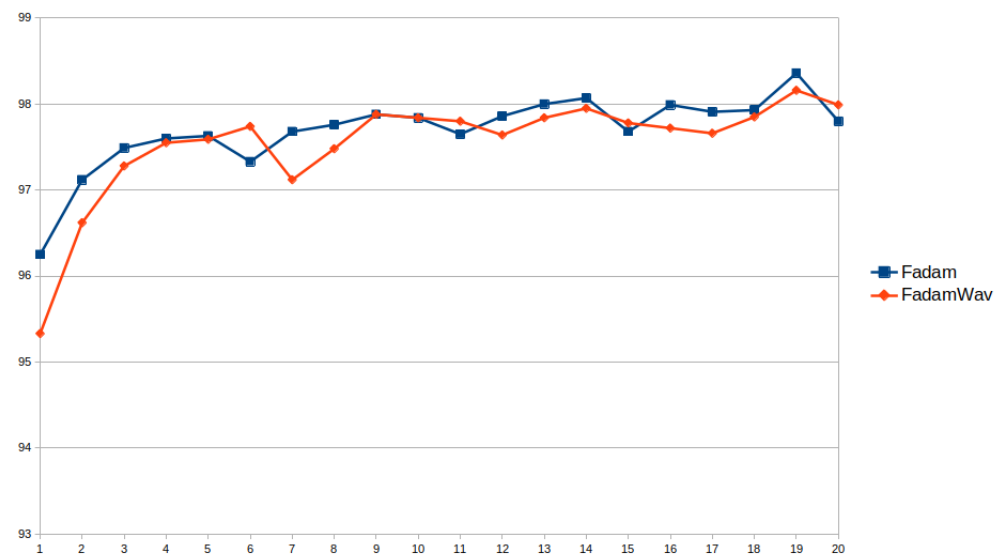


Figure 5. FAdam vs. FAdamWav with PDWT, length-four filters and $\alpha = \frac{\pi}{6}$.

3.3. Experiment 3

Experiment 3 considers FAdamWav with fractional order $\nu = 0.2$ and the parametric length-sixth filters given in Equation (20) depending on α and β . In the experiment, $\alpha_i = i \cdot \frac{\pi}{12}$ and $\beta_j = j \cdot \frac{\pi}{12}$ with $i, j \in \{1, 2, 3, 4, 5, 6\}$.

The experiments were developed in PyTorch 2.3.1, running on a GPU NVIDIA GeForce RTX 3070. The Adam optimizer used a learning rate $lr = 0.01$ and the test accuracy was used as the metric for comparison.

Figure 6 shows the accuracy of each parametric filter (α_i, β_j) . The best result is for $\alpha_i = \frac{3\pi}{12}$ and $\beta_j = \frac{5\pi}{12}$. The experiment reveals that the filter modifies the training performance of the neural network.

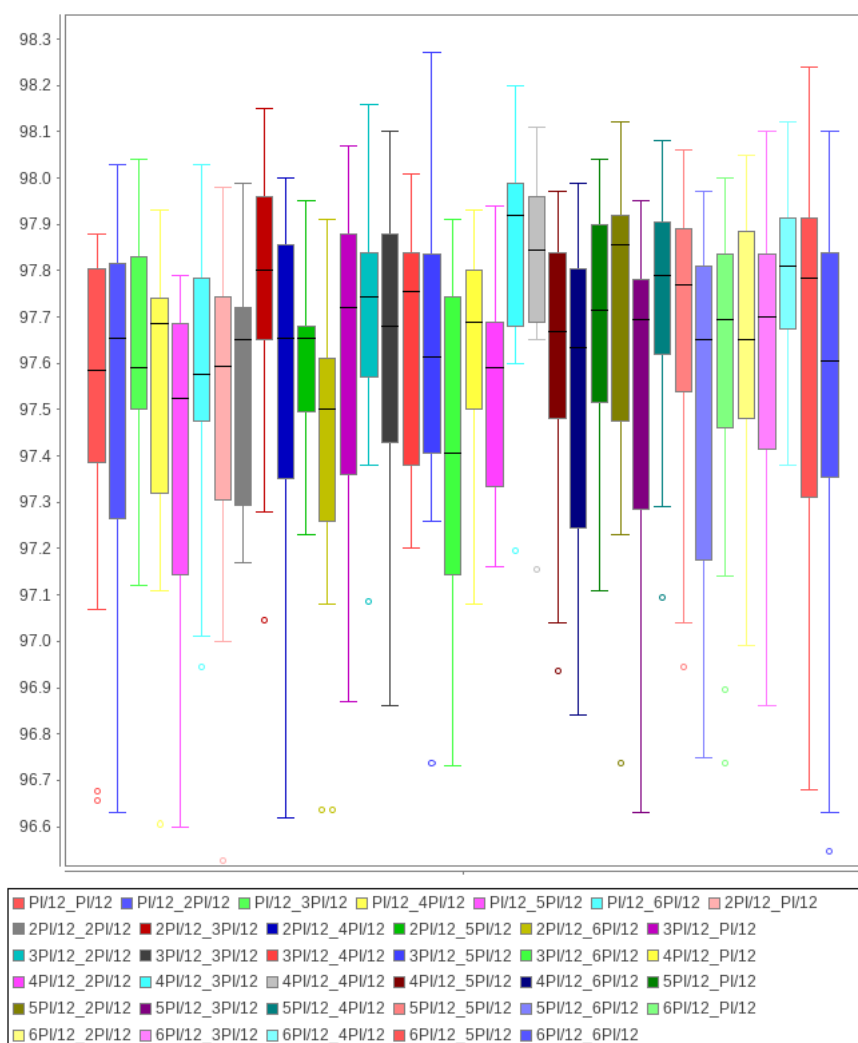


Figure 6. FAdamWav with length sixth filters, α and β equal to $j \cdot \frac{\pi}{6}$, for $j = \{1, 2, 3, 4, 5, 6\}$.

3.4. Experiment 4

In Experiment 4, the amount of GPU memory used during executions was measured. These executions developed a comparison with and without applying the wavelet transform: “FAdam” refers to the execution without wavelets whereas “FAdamWav_k1”, “FAdamWav_k2”, “FAdamWav_k3” correspond to $k = 1, 2$, and 3 wavelet transformation levels, respectively.

Note that theoretically, the memory savings are 50% for FAdamWav_k1, 75% for FAdamWav_k2, and 87.5% for FAdamWav_k3. However, as shown in Table 5, the reduction in GPU memory using wavelets does not correspond to these percentages when compared to FAdam as the reference.

Table 5. Gradient memory usage: FAdam vs. FAdamWav for $k = 1, 2$, and 3 wavelet transformation levels.

Memory Saving	FAdam	FAdamWav_k1	FAdamWav_k2	FAdamWav_k3
Theoretical (percentage)	0 MB	50% MB	75% MB	87.5%
GPU memory usage (practical measure MB)	28.8 MB	23.95 MB	18.82 MB	18.82 MB
Practical (percentage)	0 MB	16.8% MB	34.6% MB	34.6%

Nevertheless, there is a reduction in GPU memory usage: from 28.8 MB with FAdam to 23.95 MB with FAdamWav_k1, which represents a 16.8% reduction. A second transformation level reduces the GPU memory usage from 28.8 MB to 18.82 MB (a 34.6% reduction). With three transformation levels, the memory usage remains at 18.82 MB. Therefore, there is no further savings from two to three transformation levels. This is because GPU memory is assigned in blocks, so when memory usage does not change considerably, the reported memory usage remains the same.

Figure 7 shows the accuracy for FAdam (without applying wavelets), FAdamWav_k1, FAdamWav_k2, and FAdamWav_k3, during 20 epochs. In Figure 7, it is possible to appreciate that:

- FAdamWav_Level1 has a similar performance compared to FAdam, even though FAdamWav_Level1 saves 16.8% of GPU memory.
- FAdamWav_Level2 has lower accuracy than FAdam and FAdamWav_k1, but it is still competitive despite the 36.4% reduction in GPU memory.
- FAdamWav_Level3 has lower performance than FAdamWav_Level2, and no more GPU memory reduction.

The accuracies for FAdam, FAdamWav_k1, FAdamWav_k2, and FAdamWav_k3 are shown in Table 6.

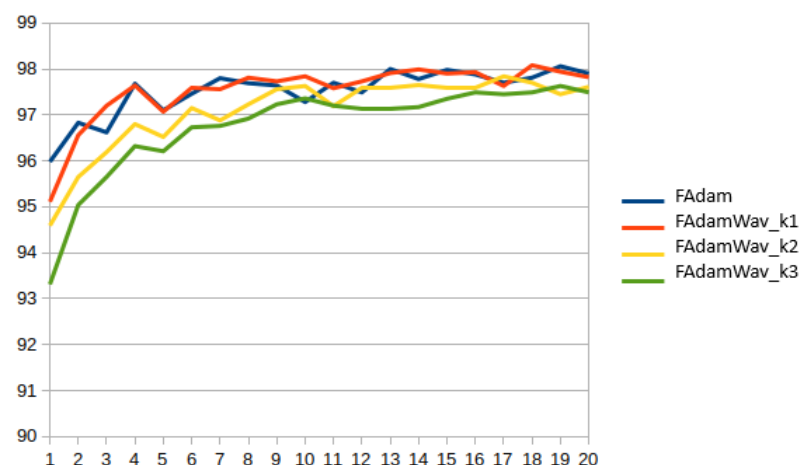


Figure 7. Adam vs. FAdamWav $k = 1, 2$, and 3 transformation levels.

Table 6. Accuracy: FAdam vs. FAdamWav for $k = 1, 2$, and 3 wavelet transformation levels.

Adam	FAdamWav_k1	FAdamWav_k2	FAdamWav_k3
95.98	95.11	94.59	93.31
96.83	96.56	95.65	95.04
96.62	97.20	96.19	96.15
97.68	97.64	96.80	96.32
97.10	97.07	96.52	96.21
97.46	97.59	97.15	96.73
97.80	97.56	96.88	96.90
97.69	97.81	97.23	96.92
97.64	97.73	97.26	97.03
97.28	97.84	97.63	97.36
97.70	97.58	97.19	97.12
97.49	97.73	97.60	97.14
98.00	97.91	97.60	97.30
97.78	97.99	97.65	97.45
97.98	97.98	97.59	97.49
97.88	97.93	97.59	97.47
97.70	97.63	97.84	97.49
97.81	98.08	97.70	97.49
98.06	97.94	97.45	97.66
97.90	97.82	97.61	97.49

The elapsed time for the 20 epochs whose data are reported in Table 6 are shown in Table 7, where it is possible to see that:

- The application of a single wavelet transformation level (FAdamWav_k1) increases the execution time for 50 seconds with respect to FAdam.
- The application of a second wavelet transformation level (FAdamWav_k2) increases the execution time for 70 seconds with respect to FAdam.
- The application of a third wavelet transformation level (FAdamWav_k3) increases the execution time to 94 seconds with respect to FAdam.

These experiments suggest that applying one or two transformation levels reduces GPU memory while maintaining competitive performance. However, using more than two transformation levels is not beneficial because it increases the execution time without providing additional memory reduction.

Table 7. Elapsed time: FAdam vs. FAdamWav for $k = 1, 2$, and 3 wavelet transformation levels.

FAdam	FAdamWav_Level1	FAdamWav_Level2	FAdamWav_Level3
131 s	181 s	201 s	225 s

4. Discussion

The optimizer is a crucial component of neural network training. Due to the rapid growth of Artificial Intelligence, several gradient-based optimizers have been proposed. The Adam optimizer is considered the state of the art in deep learning because it outperforms other optimizers by considering historical adaptations of the moments. However, a drawback is the cost of storing and computing these gradients.

In this regard, we present FAdamWav, a variant of Adam which combines wavelet-based gradient compression with fractional gradient optimization.

The main idea behind FAdamWav is to reduce the memory required for the gradients by compressing them with a discrete wavelet transform while maintaining competitive accuracy. It also aims to improve performance by leveraging the fact that the fractional gradient optimizer outperforms the integer case. The integer case is a special case of the fractional optimizer when $\nu = 1.0$.

FAdamWav is characterized by the following:

1. Wavelet gradient compression. A wavelet-based gradient compression maps the gradient matrix to a reduced space via the parametric discrete wavelet transform with k transformation levels.
2. Efficient calculation of moments and velocities. Calculating the moments m_t and velocities v_t is efficient because high-frequency bands are removed in the wavelet low-dimensional space. Therefore, the gradients do not contain high frequencies.
3. Fractional gradient optimization. The fractional ν -order of the optimizer is variable. Selecting the appropriate value of ν provides good optimization. Our experiments show that the integer case is not optimal, and that the fractional version outperforms it. For instance, in Experiment 1, the best fractional order was $\nu = 0.2$.
4. Parametric filter selection. FAdamWav allows to select optimal parameter values of the filters for the discrete wavelet transform. The goal is to achieve maximum compression so that, even when the high-frequency coefficients are set to zero, the reconstructed gradients are sufficiently similar to the original.

Although the experiments were developed by varying the parameters step by step with discrete values in the range $(0, 2\pi)$, it is possible to adjust the ν -order during execution.

5. Neural network parameter optimization. The inverse parametric discrete wavelet transform restores the gradients to their original dimensions. Once restored, the gradients are used to update the network parameters for each training step.
6. Wavelet transformation levels. FAdamWav has the possibility of applying k transformation levels, and it is possible to choose a value of k that provides an acceptable trade-off between longer execution time and lower memory usage.
7. One-dimensional discrete wavelet transform. The discrete wavelet transform is applied in one dimension. However, it is possible to deal with the gradient matrix g_t using a two dimensional wavelet transform, including curvelets or shearlets.

As previously mentioned, introducing the wavelet transform increases execution time, but this is an acceptable trade-off given the memory reduction.

Although the memory savings do not reach the theoretical limits of 50% or 75% for one and two transformation levels, the practical measurement of memory savings on the GPU is acceptable and supports the main idea behind FAdamWav.

From the experiments, it has been observed that considerable memory savings are achieved with one or two wavelet transformation levels. However, applying more than two transformation levels is not recommended. In fact, Experiment 4 shows that using three levels of the wavelet transform significantly increases execution time without reducing the amount of GPU memory.

In Experiment 2, the filters are first calculated based on the parameters, and then the DWT is applied. It is important to note that these parameters can be adjusted during training. Specifically, the parameter selection comes from a finite set of values in the range $(0, 2\pi)$. However, the filters can be modified during execution.

Our experiments only consider small models. However, it is reasonable to expect that the main idea of FAdamWav applies to more complex models such as Large Language Models (LLMs) or Convolutional Neural Networks (CNNs), which require a lot of memory. LLMs are trained on massive amounts of text data, such as ChatGPT [35], Gemini [36], or LLaMA [37], and represent a memory challenge for training. CNNs [38] are models designed to process grid-like data, such as images, for tasks like image classification, object detection, and facial recognition. When training a CNN, memory is consumed not only by the parameters but also by the gradients. Thus, they also represents a challenge for training.

In this sense, memory reduction is still an open research area, and we contribute with FAdamWav.

Our study reveals a trade-off between the execution time, the neural network accuracy, the memory reduction, and the number k of transformation levels. The experiments demonstrate this relationship and encourage further research.

Finally, based on the previous discussion, we conclude that is possible to reduce the memory required by Adam's gradients while maintaining a comparable performance to non-wavelet versions. Additionally, different parametric filters modify the training performance of a neural network, and there exists an optimal number of transformation levels that promotes the memory saving, good execution time and competitive accuracy.

Future work will focus on studying the automatic adaptation of parametric filters, the fractional order derivative, and the number of wavelet transformation levels.

Author Contributions: Conceptualization, O.H.-A., S.A.-B., J.N.-F.; Methodology, O.H.-A., S.A.-B., J.N.-F., S.R.-M. and J.A.R.-O.; Software, O.H.-A., J.A.R.-O. and S.R.-M.; Validation, O.H.-A., S.A.-B., S.R.-M., J.A.R.-O. and J.N.-F.; Writing—review and editing, O.H.-A., S.A.-B., S.R.-M., J.A.R.-O. and J.N.-F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Fractional Derivatives and the Backpropagation Update Formula for MLP

We let a training set $\{X^i, O^i\}_{i=1}^N$ with N samples and a neural network architecture be described as follows:

- X is the input layer (input data),
- H is hidden layers,
- O is the output layer,
- L is layers, $L = H + 1$ because of the hidden layers and the output layer,
- w_{kj}^l is a matrix of synaptic weights, $l \in [1, L - 1]$, that connects neuron k of layer $l + 1$ with neuron j of layer l ,
- w_{kj}^0 are synaptic weights ($l = 0$) that connect the first hidden layer with X ,
- o_{ki} is the desired output of neuron k at output layer when the i th input data is presented,
- $\varphi(x)$ is the activation function in the L layers,
- a_{ki}^L is the output of neuron k at output layer O , when the i th input data are presented and $a_k^L = \varphi(p_k^L)$ at layer O ,
- $p_k^l = \sum w_{kj}^l \cdot a_j^{l-1}$ is the potential activation of neuron k at layer l , $1 \leq l \leq L$, with inputs a_j^{l-1} . For $l - 1 = 0$, $a_j^0 = X_j$ considering the j th component of X ,
- $a_k^l = \varphi(p_k^l)$ is the output of neuron k at a hidden layer l , $1 \leq l < L$.

Note that at the output layer, the error of neuron k is $e_{ki} = a_{ki}^L - o_{ki}$ and the subindex i means that the i th input pattern is presented to the neural network.

For all the n^L neurons, the error E_i at the output layer is:

$$E_i = \frac{1}{2} \sum_{k=1}^{n^L} e_{ki}^2 = \frac{1}{2} \sum_{k=1}^{n^L} (a_{ki}^L - o_{ki})^2 \quad (\text{A1})$$

and the cumulative error of the N training samples is:

$$E = \sum_{i=1}^N E_i = \frac{1}{2} \sum_{i=1}^N \sum_{k=1}^{n^L} (a_{ki}^L - o_{ki})^2. \quad (\text{A2})$$

The backpropagation algorithm aims to find optimal values for the free parameters of the weight matrix that minimize E .

The optimization with the gradient-descent method applied to the weight updates Δw_{kj}^l is:

$$\Delta w_{kj}^l = -\eta \frac{\partial E_i}{\partial w_{kj}^l} = -\eta D_{w_{kj}^l} E_i \quad (\text{A3})$$

that points to the direction where E_i decays. In this context, $\eta > 0$ is the learning rate. Additionally, the local gradient is defined as:

$$\delta_k^l = \frac{\partial E_i}{\partial p_k^l} \quad (\text{A4})$$

and since

$$\frac{\partial E_i}{\partial w_{kj}^l} = \frac{\partial E_i}{\partial p_k^l} \cdot \frac{\partial p_k^l}{\partial w_{kj}^l} = \frac{\partial E_i}{\partial p_k^l} \cdot a_j^{l-1} = \delta_k^l a_j^{l-1} \quad (\text{A5})$$

then, Δw_{kj}^l can be expressed as:

$$\Delta w_{kj}^l = -\eta \cdot \delta_k^l \cdot a_j^{l-1}. \quad (\text{A6})$$

For $l = L$, Equation (A4) becomes δ_k^L and then, at the output layer O :

$$\delta_k^L = e_{ki} \cdot \varphi(p_k^L). \quad (\text{A7})$$

For $1 \leq l < L$, the local gradient for hidden layers is:

$$\delta_j^l = \varphi'(p_j^l) \cdot \sum_{k=1}^{n^{l+1}} \delta_k^{l+1} \cdot w_{kj}^{l+1} \quad (\text{A8})$$

and consequently, the weight updates are:

$$\Delta w_{kj}^l = -\eta \delta_k^l a_j^{l-1}. \quad (\text{A9})$$

Formulas (A1) to (A9) conform the backpropagation algorithm for neural networks. For the fractional case, the same approach for the first-order derivative $D_{w_{kj}^l} E_i$ can be used but with the fractional gradient $D_{w_{kj}^l}^\nu E_i$. In such a case, an approximation to the fractional chain rule yields to:

$$D_{w_{kj}^l}^\nu E_i = \frac{\partial E_i}{\partial w_{kj}^l} \cdot D_{w_{kj}^l}^\nu w_{kj}^l = \delta_k^l \cdot a_j^{l-1} \cdot \frac{(w_{kj}^l)^{1-\nu}}{\Gamma(2-\nu)}. \quad (\text{A10})$$

References

- Haykin, S.S. *Neural Networks and Learning Machines*, 3rd ed.; Pearson Education: Upper Saddle River, NJ, USA, 2009.
- Robbins, H.; Monro, S. A stochastic approximation method. *Ann. Math. Statist.* **1951**, *22*, 400–407.
- Lydia, A.; Francis, S. Adagrad—An Optimizer for Stochastic Gradient Descent. *Int. J. Inf. Comput. Sci.* **2019**, *6*(5), 566–568.
- Zeiler, M. ADADELTA: An adaptive learning rate method. *arXiv* **2012**, arXiv:1212.5701.
- Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* 2014, arXiv:1412.6980.
- Hu, E.J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; Chen, W. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv* **2021**, arXiv:2106.09685. <https://doi.org/10.48550/arXiv.2106.09685>.
- Zhao, J.; Zhang, Z.; Chen, B.; Wang, Z.; Anandkumar, A.; Tian, Y. Galore: Memory-efficient LLM Training by Gradient Low-rank Projection. *arXiv* **2024**, arXiv:2403.03507. <https://doi.org/10.48550/arXiv.2403.03507>.
- Wang, Y.; Yang, Z.; Chen, B.K.; Pu, F.; Li, B.; Gao, T.; Kawaguchi, K. Memory-Efficient LLM Training by Various-Grained Low-Rank Projection of Gradients. *arXiv* **2025**, arXiv:2505.01744.
- Wen, Z.; Luo, P.; Wang, J.; Deng, X.; Zou, J.; Yuan, K.; Sun, T.; Li, D. Wavelet Meets Adam: Compressing Gradients for Memory-Efficient Training. *arXiv* **2025**, arXiv:2501.07237.
- Daubechies, I. *Ten Lectures on Wavelets*; Society for Industrial and Applied Mathematics (SIAM): Philadelphia, PA, USA, 1992.
- Mallat, S. *A Wavelet Tour of Signal Processing: The Sparse Way*, 3rd ed.; Academic Press, Inc.: Cambridge, MA, USA, 2008.
- Herrera Alcántara, O.; González Mendoza, M. Optimization of Parameterized Compactly Supported Orthogonal Wavelets for Data Compression. In *Proceedings of the Advances in Soft Computing*; Batyrshin, I., Sidorov, G., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 510–521.
- Lai, M.J.; Roach, D. Parameterizations of Univariate Orthogonal Wavelets With Short Support. In *Approximation Theory X: Wavelets, Splines, and Applications*; Chui, C.K., Schumaker L.L., Stoeckler J., Eds.; Vanderbilt University Press, Nashville, TN, USA, 2001; pp. 1–10.
- Roach, D.W. A Subclass of the Length 12 Parameterized Wavelets. In *Proceedings of the Approximation Theory XIII: San Antonio 2010*; Neamtu, M., Schumaker, L., Eds.; Springer: New York, NY, USA, 2012; pp. 263–275.
- Roach, D. The complete length sixteen parametrized wavelets. *Sampl. Theory Signal Process. Data Anal.* **2025**, *23*, 12. <https://doi.org/10.1007/s43670-025-00109-0>.

16. Podlubny, I. Chapter 2—Fractional Derivatives and Integrals. In *Fractional Differential Equations*; Podlubny, I., Ed.; Mathematics in Science and Engineering; Elsevier: Amsterdam, The Netherlands, 1999; Volume 198, pp. 41–119. [https://doi.org/10.1016/S0076-5392\(99\)80021-6](https://doi.org/10.1016/S0076-5392(99)80021-6).
17. Miller, K.S.; Ross, B. *An Introduction to the Fractional Calculus and Fractional Differential Equations*; Wiley-Interscience: Hoboken, NJ, USA, 1993.
18. Bao, C.; Pu, Y.; Zhang, Y. Fractional-Order Deep Backpropagation Neural Network. *Comput. Intell. Neurosci.* **2018**, *2018*, 7361628. <https://doi.org/10.1155/2018/7361628>.
19. Wang, J.; Wen, Y.; Gou, Y.; Ye, Z.; Chen, H. Fractional-order gradient descent learning of BP neural networks with Caputo derivative. *Neural Netw.* **2017**, *89*, 19–30. <https://doi.org/10.1016/j.neunet.2017.02.007>.
20. Herrera-Alcántara, O. Fractional Derivative Gradient-Based Optimizers for Neural Networks and Human Activity Recognition. *Appl. Sci.* **2022**, *12*, 9264. <https://doi.org/10.3390/app12189264>.
21. Herrera-Alcántara, O.; Castelañ-Aguilar, J.R. Fractional Gradient Optimizers for PyTorch: Enhancing GAN and BERT. *Fractal Fract.* **2023**, *7*, 500. <https://doi.org/10.3390/fractalfract7070500>.
22. Herrera Alcántara, O. On the Best Evolutionary Wavelet Based Filter to Compress a Specific Signal. In *Proceedings of the Advances in Soft Computing*; Sidorov, G., Hernández Aguirre, A., Reyes García, C.A., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 394–405.
23. Herrera-Alcántara, O.; González-Mendoza, M. Inverse formulas of parameterized orthogonal wavelets. *Comput. Informat. Numer. Comput.* **2018**, *100*, 715–739.
24. Luchko, Y. Fractional Integrals and Derivatives: “True” versus “False”. *Mathematics* **2023**, *11*, 3003. <https://doi.org/10.3390/books978-3-0365-0494-0>.
25. Wilson, H.; Sircar, S.; Shukla, P. *Viscoelastic Subdiffusive Flows: Theory and Computation*; Fluid Mechanics and Its Applications; Springer: Singapore, 2024; Volume 138. <https://doi.org/10.1007/978-981-96-0135-6>.
26. Garrappa, R.; Kaslik, E.; Popolizio, M. Evaluation of Fractional Integrals and Derivatives of Elementary Functions: Overview and Tutorial. *Mathematics* **2019**, *7*, 407. <https://doi.org/10.3390/math7050407>.
27. Taylor, M.E. *Tools for PDE: Pseudodifferential Operators, Paradiifferential Operators, and Layer Potentials*; Mathematical Surveys and Monographs; American Mathematical Society: Providence, RI, USA, 2000; Volume 81.
28. Bényi, Á.; Maldonado, D.; Naibo, V. What is... a paraproduct. *Not. Am. Math. Soc.* **2010**, *57*, 858–868.
29. Christ, F.M.; Weinstein, M.I. Dispersion of small amplitude solutions of the generalized Korteweg-de Vries equation. *J. Funct. Anal.* **1991**, *100*, 87–109.
30. Conti, C.; Cotronei, M. Construction of Wavelet Filters: A Revisitation. In *Proceedings of the Mathematical and Computational Modelling, Approximation and Simulation*; Ibáñez-Pérez, M.J., Lamberti, P., Remogna, S., Sbibi, D., Eds.; Springer: Cham, Switzerland, 2025; pp. 3–21.
31. Candes, E.J.; Donoho, D.L. Curvelets: A Surprisingly Effective Nonadaptive Representation for Objects with Edges. In *Proceedings of the International Conference on Curves and Surfaces*, Saint-Malo, France, 1–7 July 1999; pp. 1–10.
32. Kutyniok, G.; Labate, D. (Eds.) *Shearlets: Multiscale Analysis for Multivariate Data*, 1st ed.; Applied and Numerical Harmonic Analysis; Springer: New York, NY, USA, 2012; p. 345.
33. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*; Curran Associates, Inc.: Red Hook, NY, USA, 2019; pp. 8024–8035.
34. Deng, L. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Process. Mag.* **2012**, *29*, 141–142.
35. OpenAI. ChatGPT. Large Language Model. 2023. Available online: <https://chat.openai.com/> (accessed on 23 February 2026).
36. DeepMind, G. Gemini. Large Language Model Developed by Google DeepMind. 2023 Available online: <https://deepmind.google/technologies/gemini/> (accessed on 23 February 2026).
37. Grattafiori, A.; Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Vaughan, A.; et al. The LLaMA 3 Herd of Models. *arXiv* **2024**, arXiv:2407.21783.
38. Montesinos, L.O.A.; Montesinos, L.A.; Crossa, J. Convolutional Neural Networks. In *Multivariate Statistical Machine Learning Methods for Genomic Prediction*; Springer: Cham, Switzerland, 2022; pp. 533–577. https://doi.org/10.1007/978-3-030-89010-0_13.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.