Adaptive Activation Functions Using Fractional Calculus

Julio Zamora-Esquivel, Adan Cruz Vargas, Jose Rodigo Camacho-Perez
Paulo Lopez Meyer, Hector Cordourier, Omesh Tickoo.
Intel Labs, Mexico
julio.c.zamora.esquivel@intel.com

Abstract

We introduce a generalization methodology for the automatic selection of the activation functions inside a neural network, taking advantage of concepts defined in fractional calculus. This methodology enables the neural network to define and optimize its own activation functions during the training process, by defining the fractional order of the derivative of a given primitive activation function, tuned as an additional training hyper-parameter. By following this approach, the neurons inside the network can adjust their activation functions, e.g. from MLP to RBF networks, to best fit the input data, and reduce the output error. The result show the benefits of using this technique implemented on a ResNet18 topology by outperforming the accuracy of a ResNet100 trained with CIFAR10 reported in the literature.

1. Introduction

Emergence of Deep Learning (DL) has led to multiple Neural Network (NN) topologies and architectures being proposed to solve classification problems; an important part of the definition of these architectures consists in the proper selection of the activation function from a set of commonly known options, e.g. Tanh, Sigmoid, RBF, ReLU, Softplus, etc. This set of popular options for activation functions is in constant growth due to the active effort of the Artificial Intelligence community. The usual practice is to manually select the activation function, relying heavily on the experience of the NN architect. Therefore, it is not rare to go through an exhaustive trial and error methodology, retraining the NN with a different set of activation functions in search for an optimal configuration. The available literature describes in detail the strengths and weaknesses of different activation function options. For instance, the step function is the cheapest approach computationally, but cannot be used for training since it cannot be derived through the backprogapation process [22]. Additionally, the ReLU units can be fragile during training and can "die" (according to CS231[20][12]), which means, quoting the source, that "a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any data point again". To fix this issue, the ELUs were proposed in [2]. But also the Leaky ReLUs [17], the PReLU [6], and the Swish activation function have been suggested to avoid this problem. Most of the literature explore the use of different activation functions by performing different experiments and comparing results, and in many cases the selected activation function applies for all the network, or at least, for all the units in a layer within a NN. In this work, instead of the manual exploration of the activation function that fits better, a generalized activation function is used that morphs, smoothly switching from different activation shapes, and adjusting to the shape that minimizes the error. This work represents an effort to define a fractional derivative order value (as an additional training hyper-parameter) to adjust the activation functions. allowing the automatic selection of an activation function for optimal results. Rest of this paper is organized as follows: in Section 2, a detailed description on the concepts of fractional derivative and integral is presented in order to set the theoretical bedrock of this proposed work; Section 3 explains the methodology followed using known activation functions, and how their behaviors change through fractional derivatives/integrals. The results and discussion, and our drawn conclusions are presented in Section 4 and 5, respectively.

2. Related Work

There is plenty of work in the research community related to new activation functions, showing improvements in some particular problems. This work is not introducing a new type of activation function. Instead, it tries to group existing activation functions into families, and automate the effort of selecting them properly during the architecture definition of the NN. There are currently some proposals for parametrizing the activation functions, to perform certain level of adjustment. Some examples are the Parametric Rectified Linear Unit (PReLU [6]) in which a portion of the activation function can be adjusted to create a family of leaky activation functions, or the Scaled Exponential Linear Units (SELU) [13] that adjust also a portion of the activation function using $selu(x) = \alpha e^x - \alpha$, $\alpha \in \Re$ is the trainable parameter. Most of this reported works brings the modification to other type of activation function, adding parameters to create new empirical models. Our work instead identified that most of the common activation function types can be considered a part of a family of functions, which can be generated using fractional derivatives of a primitive function by changing its derivative order. This brings the need of introducing some concepts of fractional calculus.

3. Fractional Calculus

This section describes the motivation behind the usage of fractional derivatives and integrals, as a way to define a numerical trainable hyper-parameter value to automatically select an optimal activation function in a NN.

3.1. Fractional Derivative

The concept of the derivative, in its more basic definition, is associated to natural values (i.e. the first derivative, second derivative, etc.), and can be defined as:

$$y' = \frac{\mathrm{d}y}{\mathrm{d}x}, y'' = \frac{\mathrm{d}^2y}{\mathrm{d}x}, y''' = \frac{\mathrm{d}^3y}{\mathrm{d}x}, \tag{1}$$

where the equation terms represent the first, second, and third order derivative respectively. However, is it possible to have a non-integer derivative, for example a 1.5 derivative? In recent years, fractional calculus is gaining more popularity since it provides successful aid into modeling complex dynamics [23], understanding wave propagation [10], quantum physics[15], and other applications. In order to understand how a fractional derivative works, a simple example can be useful. Here, we show that the natural n-derivatives of the function $f(x) = x^k$ are defined as:

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = kx^{k-1}, \tag{2}$$

$$\frac{\mathrm{d}^2f(x)}{\mathrm{d}x^2} = k(k-1)x^{k-2}, \tag{3}$$

$$\frac{\mathrm{d}^af(x)}{\mathrm{d}x^a} = k(k-1)(k-2)\cdots(k-a+1)x^{k-a}. \tag{4}$$

Recalling on the concept of factorial operation (!), equation 4 can be rewritten as:

$$\frac{\mathrm{d}^af(x)}{\mathrm{d}x^a} = \frac{k!}{(k-a)!}x^{k-a}, \tag{5}$$

For the case above, the factorial operator (!) can only be defined for non-negative integer numbers. In order to generate a fractional derivative, the factorial operator can be replaced by the Gamma function Γ as proposed in [16]:

$$\Gamma(z) = \int_0^\infty t^{(z-1)}e^{-t}dt, \tag{6}$$

For the particular case of $n \in \mathbb{N}$:

$$\Gamma(n) = (n-1)!, \tag{7}$$

A known efficient method to compute Gamma is [3]:

$$\Gamma(z) = \frac{e^{-\gamma z}}{z}\prod_{k=1}^{\infty}\left(\left(1+\frac{z}{k}\right)^{-1}e^{\frac{z}{k}}\right), \tag{8}$$

where γ is the Euler-Mascheroni constant ($\gamma = 0.57721..$) [1]. Thus, replacing the factorial in equation 5 by the Gamma function, the fractional derivative is then given by [9]:

$$D^af(x) = \frac{\mathrm{d}^af(x)}{\mathrm{d}x^a} = \frac{\Gamma(k+1)}{\Gamma(k+1-a)}x^{k-a}. \tag{9}$$

The above definition represents the fractional derivative of function $f(x) = x^k$ valid for $k, x \geq 0$.

3.2. Fractional Integral

Analogous to the derivative, a given function can be integrated successively. For example, using the function $y = 1$, the successive integrals J of y are given by:

$$Jy = \int_0^x dt_1 = x, \tag{10}$$

$$J^2y = \int_0^x\int_0^{t_2} dt_1dt_2 = \frac{x^2}{2}, \tag{11}$$

$$\vdots \tag{12}$$

$$J^ny = \int_0^x\cdots\int_0^{t_2} dt_1dt_2\cdots dt_n = \frac{x^n}{n!}. \tag{13}$$

Replacing the factorial operator by the Gamma function, the fractional integral for $y = 1$ is:

$$J^ay = \frac{x^a}{\Gamma(a+1)}. \tag{14}$$

In order to prove how the fractional integral is defined for any function $f(x)$, let us start with the definition of the integral as:

$$J^1f(x) = D^{-1}f(x) = \int_0^x f(t)dt, \tag{15}$$

Using the Cauchy formula for repeated integration:

$$D^{-n}f(x) = \frac{1}{(n-1)!}\int_0^x (x-t)^{n-1}f(t)dt. \tag{16}$$

Again, the Gamma function can be used to remove the factorial function and generate a fractional integral as:

$$D^{-a}f(x) = \frac{1}{\Gamma(a)} \int_0^x (x-t)^{a-1} f(t)dt, \tag{17}$$

The resulting equation is known as the Riemann-Liouville integral [18]. For example, the fractional integral of function $f(x) = 1$ using 17 results in:

$$D^{-a}f(x) = \frac{1}{\Gamma(a)} \int_0^x (x-t)^{a-1} dt, \tag{18}$$

Solving the integral:

$$D^{-a}f(x) = \frac{1}{\Gamma(a)} \left[-\frac{(x-t)^a}{a} \right]_0^x, \tag{19}$$

Evaluating the limits $(0, x)$ and recalling that $a\Gamma(a) = \Gamma(a+1)$:

$$D^{-a}f(x) = \frac{1}{\Gamma(a)} \left(\frac{x^a}{a} \right) = \frac{x^a}{\Gamma(a+1)}, \tag{20}$$

This result matches the result obtained in equation 14, which demonstrates that fractional derivatives or fractional integrals can be used depending on the primitive function selected. In the following sections, these fractional calculus concepts are used to show how some activation functions are related to each other, and how a generalized activation function represents a fractional model for the neuron.

4. Generalizing the Activation Function

There have been several activation functions proposed in the literature for usage in different NNs. However, we propose to group them into a smaller number of families by analyzing the activation functions that can be generated mathematically from other ones using Fractional Calculus. For instance, we can prove that it is possible to group, multi-quadratic and the step function into the same family, because you can choose one and generate the other by computing its fractional derivative. Based on this, we will detail three function families that can be put together in this sense.

4.1. Rectified Linear Unit (ReLU)

ReLU function proposed in [19] is the simplest non-linear activation, known for faster training processing for large network development. As opposite to the Sigmoid function, the squeezing effect of back-propagated errors is not present. ReLU is defined as Max(0,x) or:

$$f(x) = \begin{cases} x & x > 0 \\ 0 & x \leqslant 0 \end{cases} \tag{21}$$

Using the fractional derivative, a generalized activation function can be generated. In this case, the generalized function g is the $a-th$ fractional derivative of f, basically $g(x) = D^a f(x)$ using 21:

$$g(x) = \begin{cases} D^a x & x > 0 \\ 0 & x \leqslant 0 \end{cases} \tag{22}$$

where the fractional derivative of x is generated using equation 9 with $k = 1$:

$$g(x) = D^a f(x) = \frac{\Gamma(2)}{\Gamma(2-a)} x^{1-a}, \tag{23}$$

Here, $\Gamma(2) = 1! = 1$, then, the generalized ReLU case is given by:

$$g(x) = \frac{x^{1-a}}{\Gamma(2-a)}, \tag{24}$$

The generalized activation function $g(x)$ can change its shape, going from ReLU when $a = 0$, to multi-quadratic ($a = 1/2$), and to step function for $a \to 1$, as shown in Figure 1.

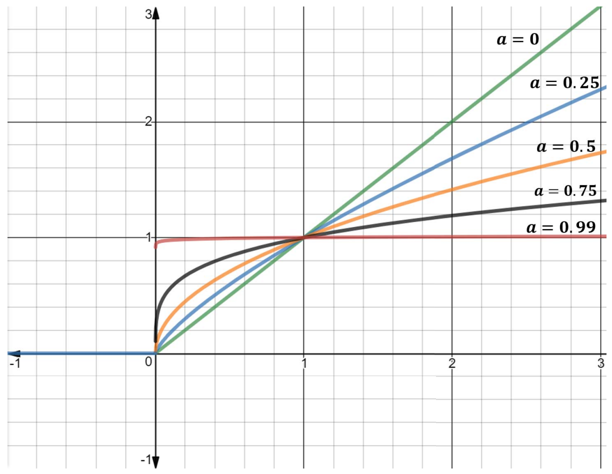

Figure 1. The fractional derivative of ReLU with order $a-th$ in the range of $[0, 1]$. Depending of the fractional derivative order a, this activation function behaves as ReLU, as multi-quadratic or as step function.

4.2. Sigmoid

The Sigmoid is a special case of the logistic function named in 1844 by Pierre François Verhulst[21]. It is called Sigmoid because it has the shape of the letter "s" and has been widely used in NNs. On the other hand, the Softplus activation function $f(x) = log(1 + e^x)$ was introduced in 2001[4] as a soft, continuous version of the ReLU function. Softplus is also the primitive function of Sigmoid, and exposes not only a different range of output values $(0, \infty)$, but also additional convex properties. In this way, the Softplus function can be used as primitive to generalize Sigmoid-like functions using fractional derivatives, with ln as particular case of log.

$$g(x) = D^a ln(1 + e^x), \tag{25}$$

then the fractional derivative of Softplus is computed as:

$$g(x) = \lim_{h \to 0} \frac{1}{h^a} \sum_{n=0}^{\infty} (-1)^n \frac{\Gamma(a+1) \ln\left(1 + e^{(x-nh)}\right)}{\Gamma(n+1)\Gamma(1-n+a)} \quad (26)$$

This generalized activation function can morph its shape from Softplus, when the fractional derivative order a is zero ($a = 0$), to Sigmoid when $a = 1$, or to a bell like shape with ($a = 2$) as illustrated in Figure 2.

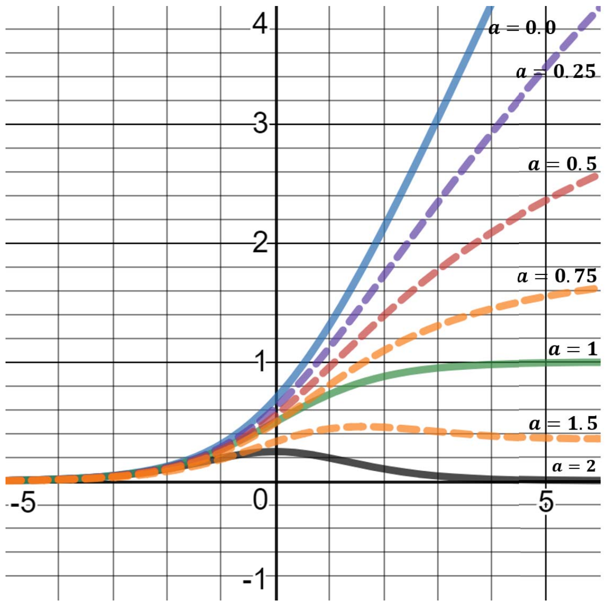

Figure 2. Fractional derivative of the Softplus Activation function with order $a-th$ in the range of $[0, 2]$. Depending on the fractional derivative order a, this activation function behaves as Softplus ($a = 0$), Sigmoid function ($a = 1$), or a bell like shape function suitable for RBF ($a = 2$).

4.3. Hyperbolic Tangent

The Hyperbolic Tangent $f(x) = Tanh(x)$ is one of the most used activation functions, mainly because is within the rage of $(-1, 1)$. Analogously to the cases shown before, the generalized Hyperbolic Tangent activation function $g(x) = D^a Tanh(x)$ can morph from Hyperbolic Tangent to $Sech^2$. In this case, the $Sech^2$ behaves similar to the activation function $y = e^{-x^2}$ used in RBF NNs. In this sense, if the generalized function g is defined like

$$g(x) = D^a \tanh(x), \quad (27)$$

similar to the Softplus case, the fractional derivative of the Hyperbolic Tangent is given by:

$$g(x) = \lim_{h \to 0} \frac{1}{h^a} \sum_{n=0}^{\infty} (-1)^n \frac{\Gamma(a+1) \tanh(x - n \cdot h)}{\Gamma(n+1)\Gamma(1-n+a)}. \quad (28)$$

Equation 28 represents a generalized activation function that produces a family of functions that morph from Hyperbolic Tangent ($a = 0$) to square hyperbolic secant function ($a = 1$) as shown in Figure 3.

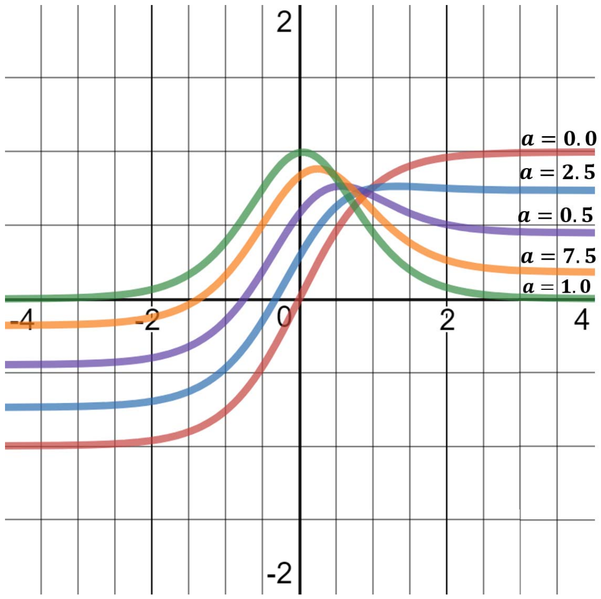

Figure 3. The fractional derivative of the Hyperbolic Tangent with order $a-th$ in the range of $[0, 1]$. Depending on the fractional derivative order a, this activation function behaves as Hyperbolic Tangent ($a = 0$), or as Square Hyperbolic Secant function ($a = 1$) used in RBF.

In Figure 3, the fractional derivative for a values in the range of $[0, 1]$ is displayed, although there is not restriction in the use of just this range of values. For example, using the $a = -1$ generates the Hyperbolic Tangent primitive function.

In conclusion, having the flexibility to morph from one activation function to another by changing a single fractional derivative order parameter enables the network to adjust the activation function to better fit the specific problem in hand, avoiding the need of handcrafted recipes from the NN architect, opening the possibility of increasing the performance due to a better optimization of the architecture.

5. Learning the Activation function

In the previous section the activation functions were grouped as a generalized Activation function, With this, the training rules required to optimize for $a-th$ order of derivative will be introduced. Using these training rules, the NN training process will adjust the parameter "a", selecting the activation function shape that minimizes the error.

5.1. Learning ReLU

A way to minimize the error is adjusting the parameter a using gradient descendent, this process requires the computation of the partial derivative of the activation function, in this case given by the equation 24:

$$\frac{\partial}{\partial a}g(x) = \frac{-x^{1-a}}{\Gamma(2-a)^2}\frac{\partial}{\partial a}\Gamma(2-a) + \frac{x^{1-a}ln(x)}{\Gamma(2-a)}\frac{\partial}{\partial a}(1-a) \tag{29}$$

Grouping the terms:

$$\frac{\partial}{\partial a}g(x) = -\left[\frac{\Gamma'(2-a)}{\Gamma(2-a)^2} + \frac{ln(x)}{\Gamma(2-a)}\right]x^{1-a} \tag{30}$$

Using equation 24 to simplify 30:

$$\frac{\partial}{\partial a}g(x) = -\left[\frac{\Gamma'(2-a)}{\Gamma(2-a)} + ln(x)\right]g(x) \tag{31}$$

where, Γ' represents the partial derivative of Γ:

$$\Gamma'(2-a) = \frac{\partial}{\partial a}\Gamma(2-a) = -\int_0^\infty t^{(1-a)}ln(t)e^{-t}dt \tag{32}$$

Using the definition of the digamma function ψ [9] :

$$\psi(z+1) = \frac{\Gamma'(z+1)}{\Gamma(z+1)} = -\gamma + \sum_{n\geq 1}^{\infty}\left(\frac{z}{n(n+z)}\right) \tag{33}$$

where γ is one of the many definitions of the Euler-Mascheroni constant ($\gamma = 0.57721..$). For $z = 0$ we have:

$$\psi(1) = \Gamma'(1) = -\gamma. \tag{3}$$

Then, the digama function of $(2-a)$ is given by:

$$\psi(2-a) = -\gamma + \sum_{n=1}^{\infty}\left(\frac{1-a}{n^2+n-an}\right) \tag{34}$$

Using this Digama function ψ, equation 31 can be rewritten as:

$$\frac{\partial}{\partial a}g(x) = -\left[\psi(2-a) + ln(x)\right]g(x) \tag{35}$$

The equation above is used to minimize the NN error by adjusting the parameter a, which means adjusting the activation function for lower error performance.

5.2. Learning Sigmoid

As detailed previously, the generalized Softplus can generate three activation functions: Softplus, Sigmoid, and RBF functions, depending on the fractional derivative order a. Here, the training rules to tune a will be defined to get the optimal value and, in consequence, the selection of the best activation function for a given problem.

From equation 26 we define the function A containing the terms depending on a:

$$A(a) = \frac{\Gamma(a+1)}{h^a\Gamma(1-n+a)} \tag{36}$$

To reduce the error, the partial derivative of 26 is needed, and it is computed using the partial derivative of A:

$$\frac{\partial}{\partial a}g(x) = \lim_{h\to 0}\sum_{n=0}^{\infty}(-1)^n\frac{\ln\left(1+e^{(x-nh)}\right)}{\Gamma(n+1)}\frac{\partial}{\partial a}A(a) \tag{37}$$

Then, we will focus on $\frac{\partial}{\partial a}A(a)$.

$$\frac{\partial}{\partial a}A(a) = A(a)\left[\frac{\Gamma'(a+1)}{\Gamma(a+1)} - \frac{\Gamma'(1-n+a)}{\Gamma(1-n+a)} - ln(h)\right] \tag{38}$$

Using the digamma function 33, the above equation can be simplified as:

$$\frac{\partial}{\partial a}A(a) = A(a)\left[\psi(a+1) - \psi(1-n+a) - ln(h)\right] \tag{39}$$

where

$$\psi(a+1) - \psi(1-n+a) = \sum_{k=1}^{\infty}\frac{n}{(k+a)(k+a-n)} \tag{40}$$

5.3. Learning Hyperbolic Tangent

The formulation of the training rule for Hyperbolic Tangent is similar to the Sigmoid rule described in Section 5.1. In an analog way, we can assess that equation 37 is now given by:

$$\frac{\partial}{\partial a}g(x) = \lim_{h\to 0}\sum_{n=0}^{\infty}(-1)^n\frac{tanh(x-nh)}{\Gamma(n+1)}\frac{\partial}{\partial a}A(a) \tag{41}$$

Here, the partial derivative of A is computed using equation 39. This training rule morphs the $tanh$ activation function to $sech^2$, and we can integrate fractionally to obtain a ReLU-like shape.

6. Experiments, Results and Discussion

To evaluate the effectiveness of our proposed technique, two different experimental scenarios were used: a shallow NN with a small artificially created training data set to analyze and visualize the behaviour of the dynamic activation functions while training; and a ResNet18 network trained with the CIFAR10 data set [14] to quantify the effect of implementing our fractional derivative technique in a well known state-of-the-art CNN architecture.

6.1. Fractional Derivative Order Visualization

For the first experiment, we opted for a simple classification task to visualize the behaviour and the effect of adjusting the activation function during training. The data set artificially generated consists of five classes: two of them, cyan and blue, generate separate clusters or cumulus of patterns; a green class is composed by two clearly separated cumulus, one on top and one at the button; these are separated by the red class which is composed by a scattered

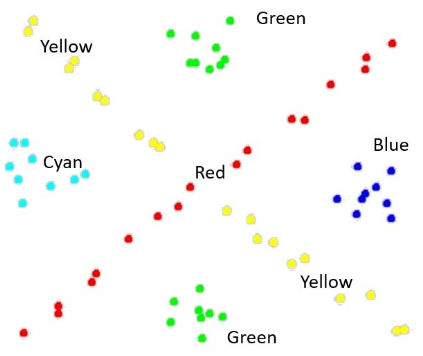

Figure 4. These artificially generated data set represents a simple but non-trivial classification problem of 5 different colors. The purpose of this data set is to provide a visualization of the generalization power presented by our proposed technique.

region of patterns; finally, a yellow class is composed by two scattered regions of patterns (see Fig 4).

The NN topology used in this experiment consists of only five units, one per class, without any hidden layers. We intentionally selected a shallow model to easily visualize the effect of training on the activation function of each neuron. A higher order NN $s = \chi^T Q \chi$ was used instead of the typical $s = \sum_n w_i \chi_i$ in order to generalize from hyper planes to quadratics as in [5]:

$$f = D^a ln(1 + e^{\chi^T Q \chi}), \tag{42}$$

where χ represents the homogeneous vector of input x, i.e. $\chi = [x, 1]$, and x is a two dimensional vector representing the input features. To compare our approach against the traditional practice of selecting a fixed activation function, we trained the model using: 1) the Softplus activation function [4], and 2) the fractional derivative of the Softplus, where the fractional derivative order a was learned as a tunable hyper-parameter. The experimental results show that not only the global error was reduced in less number of epochs when using fractional derivatives during the training, but also the error reached a lower value, as it can be seen explicitly in Figure 5.

Besides the error metric described above, it is also important to consider the generalization behavior of the trained model, to visualize that every single input vector (pattern) was evaluated, associating an output for every data sample and assigning its corresponding color. Figure 6 shows graphically the results of both models evaluated: on the left side, the fixed Softplus activation function tends to present an overfitting over all classes, i.e. set more specific boundaries over the used training data set; in contrast, our technique, shown on the right side, yields a more generalized set of boundaries.

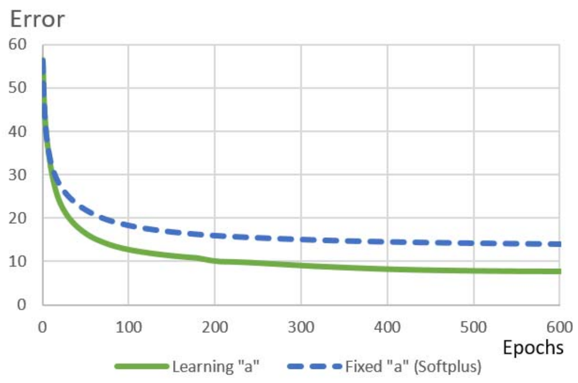

Figure 5. Error values obtained during training when Softplus activation function (dashed plot), and appied fractional derivative order a to Softplus (solid pot) were used.

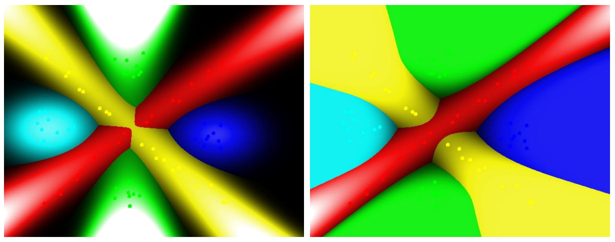

Figure 6. Graphical visualization of the obtained classification models by using Softplus activation function (left), ans Softplus activation function subject to fractional derivatives (right).

For each epoch during the training routine, we kept track of the fractional derivative order of each neuron, i.e. the type of activation function defined in each neuron. This progression can be observed in Figure 7. It can be noticed that, in the final model trained with our methodology, four of the activation functions (a_2 to a_5) morphed through the use of fractional derivatives from Softplus to Sigmoid activation function, and one (a_1) started to morph, but quickly returned to Softplus activation function, remaining like this until the end of the training process.

In traditional NN topologies, all activation functions of all the neurons inside a given layer are equally set, which means that either all Sigmoids, all Hyperbolic Tangents, or all ReLU, etc., are initially chosen. A combination of activation functions is not a common configuration, since the high number of possible combinations makes the task of training each configuration quite impractical. In contrast, our methodology is designed to automatically select the shape of each neuron's activation function that minimizes the global error. To prove this point, we experimentally observed that in our defined classification problem, by randomly initializing the fractional derivative order a in the range of $[0, 1.5]$, the model always converged to the same final configuration of activation function shapes, i.e. four

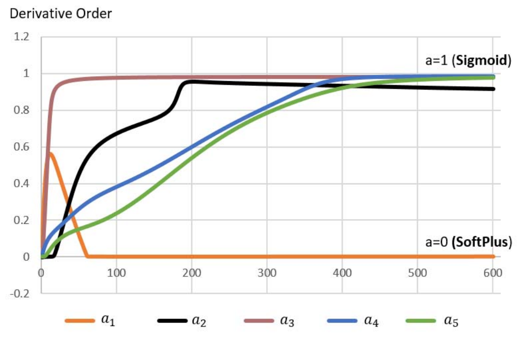

Figure 7. Evolution of the fractional derivative order a (shape of the activation function) during the training of the NN. All activation functions started with a Softplus ($a = 0$).

Sigmoid shaped and one Softlpus activation function. For instance, in Figure 8 all activation functions were initialized as Sigmoids ($a = 1$), and during training, two of them showed some shape morphing at first, but returned to Sigmoid shape as the in the experiment presented in Figure 7. Going further with the experimentation, starting from an order of $a = 2$, the method reached a different local minimum, and different units ended with different activation function shapes, as can be seen in Figure 9

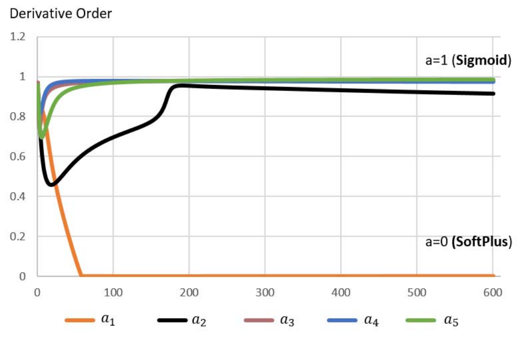

Figure 8. Evolution of fractional derivative order a (shape of the activation function) during the training of the NN. All activation functions started with a Sigmoid ($a = 1$).

6.2. ResNet18 with Trainable Activation Function

The experiments described in the previous section are useful for visualization of the training behaviour in the activation function. In this experiment, we want to measure the benefit of this technique in combination with state-of-the-art topologies, as we believe that any network can benefit from the use of this adaptive activation function using fractional derivatives. In this case, we modified a ResNet18 topology [11] to use our trainable activation function. We performed a training and testing routine using the well known CIFAR10 image database [14], and compared our findings with published results [7][8]. We have observed that the new version of ResNet18* (0.27M) over-performed the ResNet110(1.7M) in [8] with a marginal increment in the number of parameters. The performance of this network was close to ResNet50 with 25.6M parameters (94X bigger). Applying the Adaptive AF to ResNet18* (11M) it also increases 2% as shows table 1.

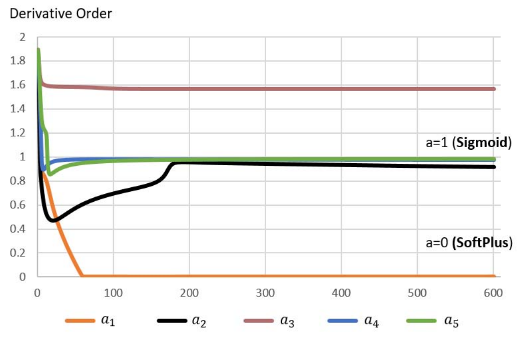

Figure 9. Evolution of fractional derivative order a (shape of the activation function) during the training of the NN. All activation functions started with a RBF ($a = 2$).

Table 1. Comparing ResNet18* (with adaptive activation function Relu)Vs ResNet18** (with adaptive activation function Adaptive Softplus) Vs reported ResNet topologies for CIFAR10

Neural Network	Depth	#Parameters	Accuracy%
ResNet18	18	0.27M	91.25
ResNet56	56	0.85M	93.03
ResNet110	110	1.7M	93.57
ResNet18**	**18**	**0.27M**	**92.92**
ResNet18*	**18**	**0.27M**	**93.57**
ResNet18	18	11M	93.02
ResNet50	50	25.6M	93.62
ResNet100	100	44.5M	93.75
ResNet18*	**18**	**11M**	**95.08**

It is important to remark that our technique does not propose a new topology, and it does not compete with other topologies or architectures. Instead, it can be combined with existing NN models to improve accuracy further, to have better generalization, or to optimize the model size.

In most of the experiments realized, the adjustments of the activation function produced a better generalization, which in consequence resulted in an increment of the testing accuracy.

6.3. ImageNet with Trainable Activation Function

The results shown in Table 2 are based on ResNet50 replacing AF with our proposed Adaptive AF. These experiments on ImageNet yield 1.06% improvement on the top-1 error. The results obtained by our modified Resnet50* over-performs other topologies not listed here like: VGG19, GoogleNet, Densenet201, Shuflenet, MobileNet.

Table 2. Comparing Error for ResNet50* (with adaptive activation function Relu) Vs reported ResNet topologies for ImageNet

Neural Network	#Param	Top1-E%	Top5-E%
ResNet18	11M	30.24	10.92
ResNet50	25.6M	23.85	7.13
ResNet101	44.5M	22.63	6.44
ResNet50*	**25.6M**	**22.79**	**6.59**

7. Conclusion

By taking advantage of the concepts defined by fractional calculus, the most commonly used activation functions, e.g. Sigmoid, ReLU, Step, Softplus, Hyperbolic Tangent, Hyperbolic Squared Secant, etc. can be grouped into families of activation functions, giving the possibility to generate sets of them by means of a fractional derivative from their main primitive function, using a new parameter a that represents the fractional derivative order. Making use of this generalized activation function (the family of the activation functions), the training rule to adjust the fractional derivative order a during backpropagation was generated. The use of this technique avoids the need of manual selection of the activation function, providing an automated optimized adjustment of each activation function. Using this methodology a single layer can mix different activation functions in order to find the best architecture for a given problem. Through our experimentation, this brings a better generalization with an increment in the accuracy when compared to the fixed activation function architectures. Applying our technique to a ResNet18 topology, the boost of accuracy was more than 2% over performing a typical ResNet100 for CIFAR10, and 1% boost for ImageNet.

References

[1] S. I. Abramowitz, M. Handbook of mathematical functions with formulas, graphs and mathematical tables. 10th ed:258–259, 1972.

[2] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *ICLR2016*, 2016.

[3] P. J. Davis. "leonhard euler's integral: A historical profile of the gamma function". *American Mathematical/ doi:10.2307/2309786*, Monthly. 66 (10):849869, 2016.

[4] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia. Incorporating second-order functional knowledge for better option pricing. In *Advances in neural information processing systems*, pages 472–478, 2001.

[5] M. M. Gupta, I. Bukovsky, N. Homma, A. M. Solo, and Z.-G. Hou. Fundamentals of higher order neural networks for modeling and simulation. In *Artificial Higher Order Neural Networks for Modeling and Simulation*, pages 103–133. IGI Global, 2013.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[9] R. Herrmann. *Fractional Calculus*. World Scientific Publishing, 2011.

[10] S. Holm and S. P. Näsholm. A causal and fractional all-frequency wave equation for lossy media. *The Journal of the Acoustical Society of America*, 130(4):2195–2202, 2011.

[11] S. R. J. S. Kaiming He, Xiangyu Zhang. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

[12] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[13] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[14] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[15] N. Laskin. Fractional schrödinger equation. *Physical Review E*, 66(5):056108, 2002.

[16] I. S. M. Abramowitz, editor. *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*, chapter 6, pages 253–266. Dover, 10th edition, 1972.

[17] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, number 30 in 1, page 3, 2013.

[18] B. Miller, Kenneth S.; Ross. *An Introduction to the Fractional Calculus and Fractional Differential Equations*. John Wiley & Sons, isbn 0-471-58884-9 edition, 1993.

[19] M. A. M. R. J. D. H. S. R Hahnloser, R. Sarpeshkar. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature 405*, pages 947–951, 2000.

[20] P. Sachin. Convolutional neural networks for image classification and captioning. *https://web.stanford.edu/class/cs231a*, 2016.

[21] P.-F. Verhulst. *Notice sur la loi que la population poursuit dans son accroissement*. Correspondance mathmatique et physique, 10 edition, 1838/Retrieved 3 December 2014.

[22] P. Werbos. Beyond regression:" new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.

[23] S. W. Wheatcraft and M. M. Meerschaert. Fractional conservation of mass. *Advances in Water Resources*, 31(10):1377–1381, 2008.