Fractional Adaptive Linear Units

Julio Zamora, Anthony D. Rhodes, Lama Nachman

Intel Labs
julio.c.zamora.esquivel@intel.com, anthony.rhodes@intel.com, lama.nachman@intel.com

Abstract

This work introduces Fractional Adaptive Linear Units (FALUs), a flexible generalization of adaptive activation functions. Leveraging principles from fractional calculus, FALUs define a diverse family of activation functions (AFs) that encompass many traditional and state-of-the-art activation functions. This family includes the Sigmoid, Gaussian, ReLU, GELU, and Swish functions, as well as a large variety of smooth interpolations between these functions. Our technique requires only a small number of additional trainable parameters, and needs no further specialized optimization or initialization procedures. For this reason, FALUs present a seamless and rich automated solution to the problem of activation function optimization. Through experiments on a variety of conventional tasks and network architectures, we demonstrate the effectiveness of FALUs when compared to traditional and state-of-the-art AFs. To facilitate practical use of this work, we plan to make our code publicly available.

Introduction

The genesis of modern Artificial Neural Networks (ANNs) can be traced to the McCulloch-Pitts neural model (Mcculloch and Pitts 1943), which provides an elegant mathematical description of the high-level functionality of a single biological neuron. In this framework, a neuron receives one or more inputs, and these inputs are then aggregated and passed through a non-linear activation function (typically a step-function). The activation function serves to approximate the "firing" mechanism of a neuron.

Remarkably, since the introduction of the McCulloch-Pitts model, very little of this basc structure has substantially changed when we compare this early vision with modern Deep Learning practices. Many of the same core operations utilized in the M-P model are also found in many other popular ANN-related models, including the early Perceptron (Rosenblatt 1958), as well as the majority of modern Deep Neural Networks (DNNs), including feed-forward networks (Hinton, Osindero, and Teh 2006; LeCun, Bengio, and Hinton 2015), Convolutional Neural Networks (CNNs) (Lecun et al. 1998; Krizhevsky, Sutskever, and Hinton 2012a; He et al. 2016a), Recurrent Neural Networks (RNNs) (Hochreiter and Schmidhuber 1997), and even more neoteric methods such as Transformers (Vaswani et al. 2017; Dosovitskiy et al. 2020), and Graph Neural Networks (GNNs) (Kipf and Welling 2017; Veličković et al. 2017).

From a conceptual perspective, DNNs are known to act as *universal function approximators*. In 1989, the first (Cybenko 1989) of several subsequent Universal Approximation Theorems (UAT) pertaining to ANNs was proven in the case of the sigmoid activation function: $\sigma(x) = \frac{1}{1+e^{-x}}$. In 1991, the UAT for ANNs was extended in a relaxed form to any bounded and non-constant activation function (Kurt and Hornik 1991); a further generalization of UAT was later applied to non-polynomial activation functions (Leshno et al. 1993).

With UAT established, the bulk of ANN research in subsequent years focused on the development of engineering and architectural improvements of ANNs to enhance the efficiency of feature processing and feature learning. These innovations included the introduction of residual connections, feature normalization, novel regularization methods, and multi-scale feature aggregation, among others (Krizhevsky, Sutskever, and Hinton 2012b; He et al. 2016b; Szegedy et al. 2015; Ioffe and Szegedy 2015; Chen et al. 2016). In large part, these historical design enhancements have ignored explicit modifications made to activation functions. This inattention is possibly due to the generalization of large families of AFs as conduits to *universal approximation* expressed by the UAT, which gives the subtle (but misguided) impression of their relative insignificance. Until 2010, practitioners almost universally employed the conventional sigmoid activation function in ANN design. However, the inherent limitations of the sigmoid activation function, exhibited most starkly by the *vanishing gradient* phenomenon (Pascanu, Mikolov, and Bengio 2013), became clear by the early 2010s as researchers pushed to increase the capacity of ANNs by introducing deeper architectures.

Today, the default activation function used for DNNs is the Rectified Linear Unit (ReLU) (Nair and Hinton 2010), defined as: $f(x) = max(0, x)$. Although it was popularized nearly a decade ago with the remarkable performance of AlexNet (Krizhevsky, Sutskever, and Hinton 2012b), it is still, for this reason, often implicitly synonymized as a fixture of "Deep Learning". The ReLU activation is commonly preferred by practitioners due to its computational simplicity, favorable non-saturation properties, and the perception

of its robustness to undesirable behavior including vanishing gradient (Lu et al. 2019).

Since the introduction of the ReLU, activation function research has been severely underemphasized in literature. Nevertheless, it is well-known that activation functions play a vital role in the performance of DNNs (Wang et al. 2020; Nwankpa et al. 2018). It is only recently that the dominance of ReLU activations has come under scrutiny (Lu et al. 2019). While some novel activation functions have been proposed to supplant ReLU, to date, few alternative activation functions have enjoyed extensive adoption in the deep learning community due to their inconsistent performance and excessive complexity.

In this work, we present a novel activation function termed *Fractional Adaptive Linear Units* (FALUs). FALUs leverage fractional calculus (Luchko 2020) to render adaptive, i.e. trainable, activation functions that encapsulate the expressivity of several of the current state-of-the-art activation functions, including the GELU (Hendrycks and Gimpel 2020) and Swish (Ramachandran, Zoph, and Le 2018) functions, in addition to a large family of interpolations and variants between these functions. Moreover, by introducing a tunable "fractional derivative" parameter, the FALU activation is additionally capable of manifesting a diverse family of traditional activation functions, including the sigmoid and Gaussian functions. In this way, FALUs capture a richness and flexibility exceeding that of other activation functions.

In the following sections we summarize related work on activation functions, provide a principled technical background for Fractional Adaptive Linear Units, and demonstrate their practical performance gains over state-of-the-art and traditional activation functions across a variety of network architectures and tasks.

Related Work

Since the widespread adoption of ReLU as the *de facto* activation function used with DNNs, most activation function research has focused on exploiting the overarching benefits presented by ReLU, including its simple step-function form, non-saturating derivative, and sparse firing rate. Innovations to activation functions are consequently often hand-designed to enhance a particular property of the ReLU function that is considered to be essential; in some instances these decisions are informed by search (Ramachandran, Zoph, and Le 2018).

(Nair and Hinton 2010) introduced the Softplus activation, the *primitive* of the sigmoid function, defined:

$$f(x) = log(1 + exp(x)) \tag{1}$$

The softplus activation function serves as a smoothed version of the ReLU, while sacrificing sparsity and computational simplicity.

Attempts to increase the expressivity of ReLUs led to the introduction of several related parametric ReLU-based activation functions. In 2013, the Leaky ReLU (Maas, Hannun, and Ng 2013):

$$f(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases} \tag{2}$$

codified a piecewise linear activation function that allows for information flow via small negative values for non-firing neuron states. The Parametric Rectified Linear Unit (PReLU) (He et al. 2015) and Exponential Linear Unit (ELU) (Clevert, Unterthiner, and Hochreiter 2016) extended the general concept of the Leaky ReLU to a family of "leaky" activations by introducing a trainable parameter that adjusts the slope/shape of the negative portion of the activation function. Similarly, the Scaled Exponential Linear Units (SELU) (Klambauer et al. 2017):

$$f(x) = \lambda \begin{cases} x & x \geq 0 \\ \alpha(exp(x) - 1) & x < 0 \end{cases} \tag{3}$$

with fixed $\alpha \approx 1.6733$ and $\lambda \approx 1.0507$, proposed a smoothed negative activation function component. Klambauer et al. show that SELU activations induce self-normalizing properties in network layers.

Despite the benefits introduced by parametric ReLU function variants, these solutions nevertheless conventionally impose concrete limitations on the activation function form either by forcing component linearity or limiting the ability to fine-tune the function.

To improve activation function flexibility, Agostinelli et al. developed Adaptive Piecewise Linear (APL) activation units (Agostinelli et al. 2014). APLs are defined as a sum of hinge-shaped functions:

$$f_i(x) = \max(0, x) + \sum_{s=0}^{S} a_i^s \max(0, -x + b_i^s) \tag{4}$$

where S is a hyperparameter corresponding with the number of hinges, and a_i^s, b_i^s for $i \in 1, \ldots, S$ are tunable parameters that control the slopes of the linear segments and the location of the hinges, respectively. APL functions sacrifice function simplicity for improved expressivity. In total, APLs require training $2 \cdot SM$ new parameters (where M is the total number of hidden units); in addition, APL function components are non-smooth.

Kernel-based Activation Functions (KAF) (Scardapane et al. 2019) model the activation function in terms of a kernel expansion over D terms:

$$f(x) = \sum_{i=1}^{D} a_i k(x, d_i) \tag{5}$$

where $\{a_i\}_{i=1}^D$ are the mixing coefficients, $\{d_i\}_{i=1}^D$ are dictionary elements, and $k(.,.)$ is a kernel function. In comparison with APL activation units and parametric ReLUs, KAFs are smooth over their entire domain and capable of approximating any continuous function. However, KAFs require the introduction of additional design choices and parameter tuning regimes due to the inclusion of kernel functions and mixing coefficients.

Hendrycks and Gimpe proposed the Gaussian Error Linear Unit (GELU), activation function:

$$f(x) = x \cdot \Phi(x) \tag{6}$$

where $\Phi(\cdot)$ represents the standard Gaussian cdf. GELU functions exemplify a smoothed ReLU shape with an

asymptotically-bounded negative region. Instead of gating inputs by their sign as in RELUs, the GELU weights inputs by the magnitude of their value. The key motivation for the GELU is that it serves as a simple regularizer. Because neuron inputs tend to follow a normal distribution following Batch Normalization, the expression $x \cdot \Phi(x)$ ensures that (small) outlier input values are "dropped", since the GELU scales input values by how much greater they are than other inputs. In practice, the authors employ the simple approximation $x \cdot \sigma(1.702x)$ for (6).

(Ramachandran, Zoph, and Le 2018) leverage automatic search techniques to discover multiple novel activation functions. Through experiments, they show that the best discovered such function, termed the Swish activation:

$$f(x) = x \cdot \sigma(\beta x) \tag{7}$$

with β a constant or trainable parameter, outperforms ReLU across a variety of models and problem types. Like ReLU, the Swish function is unbounded above and bounded below. Unlike ReLU, the Swish function is smooth and non-monotonic (preserving the value of small negative inputs). (Misra 2019) presents a closely-related successor to Swish with improved regularization properties.

The proposed method falls under the general heading of learning/adaptive activation functions that, in lieu of fixed AFs, introduce trainable activation function parameters (Dubey, Singh, and Chaudhuri 2021). These parameters allow the AF to gracefully calibrate the model with the dataset complexity, while requiring additional parameter training. Of the aforementioned AFs, the APL, PReLU, and Swish functions represent adaptive activation functions. In contrast to these previous solutions, our method automates AF tuning across a diverse family of activation functions, including previously undiscovered interpolated AFs.

Fractional Calculus

In recent years, fractional calculus has proved to be a successful tool for modeling complex dynamics (Wheatcraft and Meerschaert 2008), wave propagation (Holm and Näsholm 2011), and quantum physics (Laskin 2002; Iomin 2018), among other applications (Baleanu and Agarwal 2021). In the following section, we give a brief summary of the notion of a fractional derivative from fractional calculus. Using these concepts, we subsequently provide a formal discussion of our FALU activation function.

Fractional Derivative

Conventionally, the derivative of a function is defined over natural values (i.e. the first derivative, second derivative, etc.) and is notated:

$$y' = \frac{\mathrm{d}y}{\mathrm{d}x}, y'' = \frac{\mathrm{d}^2y}{\mathrm{d}x}, y''' = \frac{\mathrm{d}^3y}{\mathrm{d}x}, \tag{8}$$

where the above equations connote the first, second, and third order derivative, respectively. While it is less-well known, conceptually, it is possible to extend the notion of derivatives to non-integer values using fractional calculus (Ortigueira 2011).

To better understand how a fractional derivative works, we begin with a simple example. Recall that the natural n-derivatives of the power function $f(x) = x^k$ are defined as:

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = kx^{k-1}, \tag{9}$$

$$\frac{\mathrm{d}^2f(x)}{\mathrm{d}x^2} = k(k-1)x^{k-2}, \tag{10}$$

$$\frac{\mathrm{d}^\alpha f(x)}{\mathrm{d}x^\alpha} = k(k-1)\cdots(k-\alpha+1)x^{k-\alpha} \tag{11}$$

Using the factorial operation (!), equation (11) can be rewritten as:

$$\frac{\mathrm{d}^\alpha f(x)}{\mathrm{d}x^\alpha} = \frac{k!}{(k-\alpha)!}x^{k-\alpha}, \tag{12}$$

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 (M. Abramowitz 1972):

$$\Gamma(z) = \int_0^\infty t^{(z-1)}e^{-t}dt, \tag{13}$$

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

$$\Gamma(n) = (n-1)!, \tag{14}$$

A known efficient method to compute Gamma is (Davis 2016):

$$\Gamma(z) = \frac{e^{-\gamma z}}{z}\prod_{k=1}^{\infty}\left(\left(1+\frac{z}{k}\right)^{-1}e^{\frac{z}{k}}\right), \tag{15}$$

where γ is the Euler-Mascheroni constant ($\gamma = 0.57721..$) (M. Abramowitz 1972). Thus, replacing the factorial in equation 12 by the Gamma function, the fractional derivative is then given by (Herrmann 2011):

$$D^a f(x) = \frac{\mathrm{d}^\mathrm{a} f(x)}{\mathrm{d}x^a} = \frac{\Gamma(k+1)}{\Gamma(k+1-a)}x^{k-a}. \tag{16}$$

The definition above represents the fractional derivative of function $f(x) = x^k$ valid for $k, x \geq 0$. We further extend these concepts below in building our FALU activation function.

Fractional Adaptive Linear Units

As a desideratum, we wish to construct an adaptive, computationally-efficient activation function that preserves the strengths of current state-of-the-art AFs while providing enhanced expressiveness and performance.

To this end, we begin by defining Fractional Adaptive Linear Units as a dynamic generalization of the Swish activation by introducing two tunable parameters: α, a real-valued *fractional derivative*, and β, a scaling parameter:

$$f(x) = D^\alpha x\sigma(\beta x) \tag{17}$$

In particular, when $\alpha = 0$ and $\beta = 1$, the FALU yields the standard Swish function, and when $\alpha = 0$ and $\beta = 1.702$, (17) reduces to the approximated GELU activation. The

FALU thus represents a flexible activation function framework encompassing both the Swish and GELU activations (plus many more related morphologies).

Using fractional calculus, we define fractional derivatives by invoking the Gamma function, $\gamma(x)$, with $x > 0$. Let $g(x, \beta) = x\sigma(\beta x)$, the parametrized Swish introduced in (7). Formally, using fractional calculus, we define the fractional derivative of $g(x, \beta)$:

$$D^{\alpha} g(x,\beta) = \lim_{\delta \to 0} \frac{1}{\delta^{\alpha}} \sum_{n=0}^{\infty} (-1)^n \frac{\Gamma(\alpha+1) g(x - n\delta, \beta)}{\Gamma(n+1)\Gamma(1-n+\alpha)} \tag{18}$$

To generate explicit update rules (i.e., for use in backpropagation schemes) for networks using FALUs, we next calculate $\frac{\partial}{\partial \alpha} D^{\alpha} g(x, \beta)$ and $\frac{\partial}{\partial \beta} D^{\alpha} g(x, \beta)$.

To compute $\frac{\partial}{\partial \alpha} D^{\alpha} g(x, \beta)$, we isolate all factors involving the α parameter; notationally, let $A(\alpha) = \frac{\Gamma(\alpha+1)}{g^{\alpha}\Gamma(1-n+\alpha)}$. One can show that:

$$\frac{\partial}{\partial \alpha} A(\alpha) = A(\alpha)[\psi(\alpha+1) - \psi(1-n+\alpha) - ln(\delta)], \tag{19}$$

where: $\psi(\alpha+1) - \psi(1-n+\alpha) = \sum_{k=1}^{\infty} \frac{n}{(k+\alpha)(k+\alpha-n)}$. Putting this together, we get:

$$\frac{\partial}{\partial \alpha} D^{\alpha} g(x,\beta) = \lim_{\delta \to 0} \frac{1}{\delta^{\alpha}} \sum_{n=0}^{\infty} (-1)^n \frac{g(x - n\delta, \beta)}{\Gamma(n+1)} \frac{\partial}{\partial \alpha} A(\alpha) \tag{20}$$

and

$$\frac{\partial}{\partial \beta} D^{\alpha} g(x,\beta) = \lim_{\delta \to 0} \frac{1}{\delta^{\alpha}} \sum_{n=0}^{\infty} (-1)^n \frac{A(\alpha)}{\Gamma(n+1)} \frac{\partial}{\partial \beta} g(\Delta x, \beta) \tag{21}$$

where $\frac{\partial}{\partial \beta} g(\Delta x, \beta) = (\Delta x)^2 \sigma(\beta \Delta x)(1 - \sigma(\beta \Delta x))$ and $\Delta x = x - n\delta$.

While the formulas in (20) and (21) provide explicit, exact derivative formulas that can be used to update the tunable parameters in (17), they are nevertheless cumbersome for practical implementations. For this reason, we next derive computationally tractable approximations to equation (18). For simplicity, we consider the following parameter domains: $\alpha \in [0, 2]; \beta \in [1, 10]$ (see Figure 3).

In our approximation we retain only the first two terms appearing in (18). Considering the first term ($n = 0$), we have:

$$(-1)^n \frac{\Gamma(\alpha+1) g(x - n\delta, \beta)}{\Gamma(n+1)\Gamma(1-n+\alpha)} = \frac{\Gamma(\alpha+1) g(x, \beta)}{\Gamma(1)\Gamma(1+\alpha)} = g(x, \beta) \tag{22}$$

And for $n = 1$, recalling that $\Gamma(\alpha+1) = \alpha\Gamma(\alpha)$, we get:

$$-\frac{\Gamma(\alpha+1) g(x - \delta, \beta)}{\Gamma(2)\Gamma(\alpha)} = -\frac{\alpha}{2} g(x - \delta, \beta) \tag{23}$$

The factor $\frac{1}{\delta^{\alpha}}$ in equation (18) is a scalar governed by $\alpha \in [0, 1]$ and the approximation step-size δ, where we set $\delta = 0.5$. With these parameters, $\frac{1}{\delta^{\alpha}} \in [0.5, 1]$; for further simplicity, we round this factor to 1, yielding:

$$D^{\alpha} g(x,\beta) \approx g(x,\beta) - \frac{\alpha}{2} g(x - 0.5, \beta) \tag{24}$$

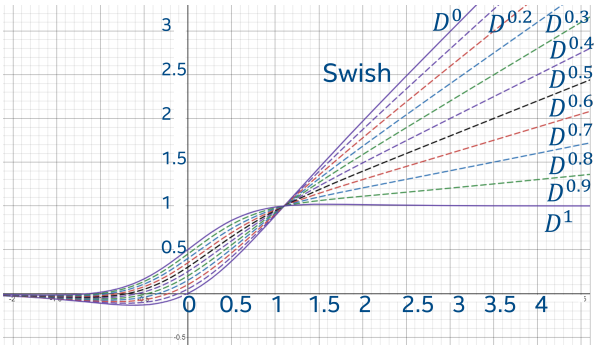

Figure 1: Family of activation functions generated by changes in the order of the derivative α in the range of $(0, 1)$ in the vicinity of $\beta = 1$.

This approximation can be used in the vicinity of $\alpha = 0$. To find the approximation in the vicinity of $\alpha = 1, \beta = 1$, we use the first derivative of the FALU. In this particular case, where $g(x, 1) = g(x) = x\sigma(x)$, $D^1 g(x, 1)$ is given by:

$$\begin{aligned} D^1 g(x) &= \sigma(x) + x\sigma(x)(1 - \sigma(x)) && (25) \\ &= x\sigma(x) + \sigma(x)(1 - x\sigma(x)) && (26) \\ &= g(x) + \sigma(x)(1 - g(x)) && (27) \end{aligned}$$

Similarly, the fractional derivative can be approximated using:

$$D^{\alpha} g(x) = g(x) + \alpha\sigma(x)(1 - g(x)) \tag{28}$$

Evaluating $\alpha = 0$ in (28) yields $D^0 g(x) = g(x)$, and evaluating for $\alpha = 1$ gives $D^1 g(x) = g(x) + \sigma(x)(1 - g(x))$, corresponding to the original Swish AF and its derivative, respectively. The family of activation functions generated by modulating this parameter is shown in Figure 1. In general, for parameter β in the equation (28), we drop the $\alpha(\beta - 1) g(x, \beta)$ term to maintain simplicity, which gives the further simplification:

$$D^{\alpha} g(x,\beta) \approx g(x,\beta) + \alpha\sigma(\beta x)(1 - g(x,\beta)) \tag{29}$$

We use (29) to approximate FALUs for $\alpha \in [0, 1]$; this family of activations is shown in Figure 3. Finally, to find the approximation of the fractional parameter $\alpha \in [1, 2]$, we compute the derivative of (28) using:

$$D^2 g(x) = D^1 \left(g(x) + \sigma(x)(1 - g(x)) \right) \tag{30}$$

Defining $h(x)$ as the first derivative of $g(x)$,

$$h(x) = D^1 g(x) = g(x) + \sigma(x)(1 - g(x)), \tag{31}$$

Equation (30) can be rewritten as:

$$D^1 h(x) = h(x) - \sigma(x)h(x) + \sigma(x)(1 - \sigma(x))(1 - g(x)) \tag{32}$$

Expanding $\sigma(x)(1 - \sigma(x))(1 - g(x))$ of (32):

$$D^1 h(x) = h(x) - \sigma(x)h(x) + \sigma(x)(1 - h(x)) \tag{33}$$

Regrouping (33):

$$D^1 h(x) = h(x) + \sigma(x)(1 - 2h(x)) \tag{34}$$

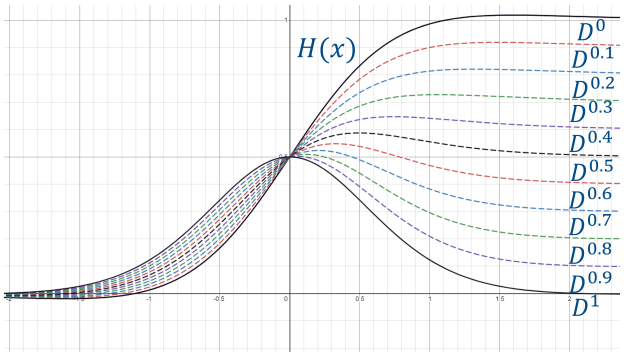

Figure 2: Family of activation function generated by changes in the order of the derivative α in the range of $(0, 1)$ in the vicinity of $\beta = 1$ for $h(x)$, or equivalently, for the derivative of $g(x)$ in the range of $(1, 2)$.

Using equation (34), we can approximate the fractional derivative of $h(x)$ for $\alpha \in [0, 1]$ as:

$$D^{\alpha}h(x) = h(x) + \alpha\sigma(x)(1 - 2h(x)) \tag{35}$$

Evaluating $\alpha = 0$ in (35) produces $D^0h(x) = h(x)$, and evaluating $\alpha = 1$, gives $D^1h(x) = h(x)+\sigma(x)(1-2h(x))$, which correspond to the first and second derivative of the Swish function, respectively. The family of activation functions rendered by changing α is shown in Figure 2. When we include the β parameter, this yields an approximation of the FALU for $\alpha \in [1, 2]$:

$$D^{\alpha}h(x, \beta) \approx h(x, \beta) + \alpha\sigma(\beta x)(1 - 2h(x, \beta)) \tag{36}$$

Together, when we combine equations (29) and (36), we arrive at a complete specification of the FALU approximation for $\alpha \in [0, 2]$ and $\beta \in [1, 10]$:

$$D^{\alpha}g(x, \beta) \approx \begin{cases} g(x, \beta) + \alpha\sigma(\beta x)(1 - g(x, \beta)), \alpha \in [0, 1] \\ h(x, \beta) + \alpha\sigma(\beta x)(1 - 2h(x, \beta)), \alpha \in (1, 2] \end{cases} \tag{37}$$

where $h(x, \beta) = g(x, \beta) + \sigma(x)(1 - g(x, \beta))$. For implementation purposes, equation (37) can be executed with backpropagation efficiently using only a few lines of code in standard automatic differentiation workflows (our code is included).

Experimental Results

To evaluate our method, we tested the FALU activation function in comparison with a large set of baseline AFs, including thr sigmoid, ReLU (Nair and Hinton 2010), ELU (Clevert, Unterthiner, and Hochreiter 2016), SELU (Klambauer et al. 2017)], KAF (Scardapane et al. 2019), PReLU (He et al. 2015), and GELU (Hendrycks and Gimpel 2020)) AFs across several standard datasets (MNIST (LeCun and Cortes 2010), Fashion-MNIST (Xiao, Rasul, and Vollgraf 2017), CIFAR-10 (Krizhevsky 2009), ImageNet (Deng et al. 2009)), and varying model architectures. For each experiment we used the Adam optimizer (Kingma and Ba 2014) to train our model, and randomly initialized the FALU parameteres in the range $\alpha \in [0, 1]$ and $\beta \in [1, 1+\epsilon]$, with $\epsilon = 0.05$; for each dataset we use conventional train/test splits used in literature. In addition, for stability purposes, the FALU function parameters were clamped during training within the domains described previously, i.e., $\alpha \in [0, 2]$ and $\beta \in [1, 10]$. We report the maximum accuracy in five trials for each experiment; where appropriate, we report model performance as provided in research literature. As we detail below, the FALU activation consistently matched or out-performed the best-performing baseline AFs in each of our experiments.

Neural Network	#Param	Top1%
6c,10c,10c,10fc + ReLU	29K	99.2
6c,10c,10c,10fc + FALU	**29K + 26**	**99.3**
6c,p,10c,10c,10fc + ReLU	5.6K	99.0
6c,p,10c,10c,10fc + FALU	**5.6K + 26**	**99.2**

Table 1: MNIST experiment comparing model accuracy for simple, compact CNN models, where 'c' denotes convolution, 'p' denotes pooling, and 'fc' is a fully-connected layer.

MNIST

The MNIST dataset is a public dataset used to train machine learning models to classify individual handwritten digits. MNIST consists of 60,000 (50k/10k train/test split) 28×28 resolution gray scale images in 10 classes, with 6,000 images per class. Today, most state-of-the-art handwriting recognition models exceed human-level performance, and even simple ANNs are sufficient to reach 99% accuracy on MNIST. For this reason, using the MNIST dataset, we aim to demonstrate the efficacy of the FALU activation in the case of extremely compact models. For our experiments, we use two different network architectures: (1) a $29K$ parameter CNN consisting of: six traditional 5×5 convolutional filters in the first layer, ten 5×5 convolutional filters in the second layer, ten 5×5 convolutional filters in the third layer, followed by a final FC layer; and (2) a ($5X$ smaller) related topology where the second convolution layer is replaced with a pooling layer, rendering a model with only $5.6K$ trainable parameters. As shown in Table 1, the use of FALU increased prediction accuracy for MNIST across these extremely compact models by 0.1% and 0.2%, respectively, when compared with the baseline ReLU AF. Figure 4 provides a histogram of α values resulting from our trained MNIST model. Notably, the model converged to a wide range of AFs, including ReLU, sigmoid, and Gaussian morphologies, plus various interpolations between these function types.

CIFAR-10

CIFAR-10 is a public dataset consisting of 60,000 (50k/10k train/test split) 32×32 resolution RGB images in 10 classes, with 6,000 images per class. For all of our CIFAR-10 experiments, we augmented the baseline dataset using horizontal flipping, padding, and 32×32 random cropping during training. We used a modified version of Resnet18 (described in Table 2) for comparison, replacing all network AFs with the FALU (this configuration is denoted ResNET18a).

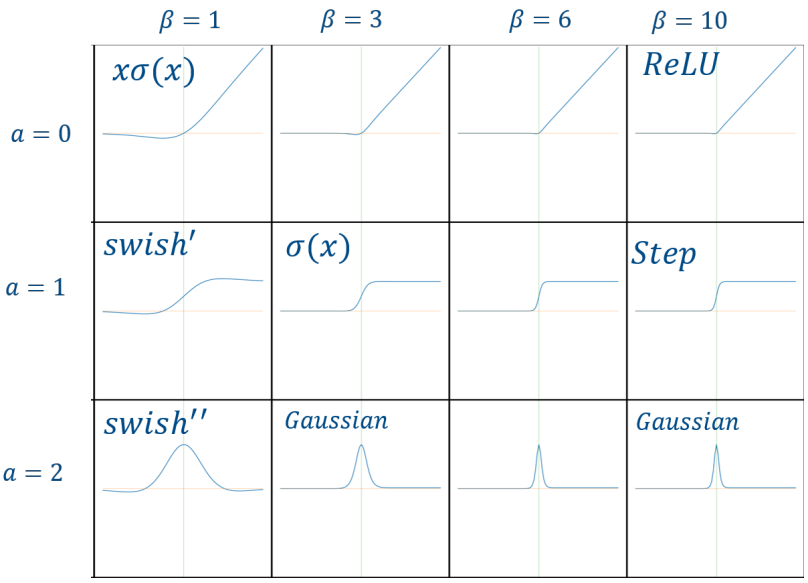

Figure 3: Family of FALU activation functions generated by evaluating the parameters $\alpha \in [0, 2]$ and $\beta \in [1, 10]$.

Name	Output Size	ResNet-18a
Conv1	$32 \times 32 \times 16$	3×3, 16 stride 1
Conv2	$32 \times 32 \times 16$	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix} \times 2$
Conv3	$16 \times 16 \times 32$	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 2$
Conv4	$8 \times 8 \times 64$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
Average pooling	$1 \times 1 \times 64$	8×8
Fully-Connected	10	64×10
Softmax	10	

Table 2: Overview of the compact ResNet18a model topology used for our CIFAR-10 experiments. ResNet18a consists of $[16, 32, 64]$ filter depths with $0.27M$ total parameters; the related ResNet50a and ResNet100a compact models are defined similarly.

Neural Network	Depth	#Param	Error%
All-CNN (Springenberg et al. 2014)	9	1.3M	7.25
MobileNetV1 (Howard et al. 2017)	28	3.2M	10.76
MobileNetV2 (Sandler et al. 2018)	54	2.24M	7.22
ShuffleNet 8G (Zhang et al. 2018)	10	0.91M	7.71
ShuffleNet 1G (Zhang et al. 2018)	10	0.24M	8.56
HENet (Qiuyu Zhu 2018)	9	0.7M	10.16
ResNet18a + ReLU (Kaiming He and Sun 2015)	20	0.27M	8.75
ResNet18a + FALU	20	**0.27M**	**7.36**

Table 3: CIFAR-10 Classification error vs number of parameters, for common compact model architectures vs. ResNet18a + FALU.

Table 3 summarizes the effect of applying FALU with the compact variant of ResNet18 (0.27M trainable parameters, see Table 2) on CIFAR-10. When compared with the identical model topology using the (default) ReLU activation, FALU yields a 1.39% error reduction. In addition, Table 3 lists results for CIFAR-10 on related compact baseline models. Despite using between $5X$-$10X$ fewer parameters, the FALU-based ResNet18 performs comparably with the best performing of these compact models.

In Table 4 we report results for several compact and large-scale ResNet topologies on CIFAR-10 (compact model variants are denoted with an 'a'), across several baseline AFs including ELU, SELU, KAF, ReLU, PReLU, and GELU. ResNet18a (see Table 2) consists of $[16, 32, 64]$ filter depths with $0.27M$ total parameters; the ResNet50a topology uses $[3, 4, 6, 3]$ block sizes, and ResNet100a consists of $[3, 4, 23, 3]$ block sizes, respectively; ResNet18b utilizes $[64, 128, 256]$ filter depths for a total of $4.29M$ parameters. In each experiment, the FALU activation function outperformed each of the baselines AFs, including the state-of-the-art GELU function. Notably, the performance gains exhibited by FALU over baseline AFs were more appreciable for larger model sizes.

Neural Network	Depth	#Parameters	Acc.%
ResNet18a + ELU	18	0.27M	91.09
ResNet18a + SELU	18	0.27M	91.09
ResNet18a + KAF	18	0.27M + 6080	91.18
ResNet18a + ReLU	18	0.27M	91.25
ResNet18a + PReLU	18	0.27M + 19	92.29
ResNet18a + GELU	18	0.27M	92.56
ResNet18a + FALU	**18**	**0.27M + 688**	**92.64**
ResNet50a + ReLU	56	0.85M	93.03
ResNet100a + ReLU	110	1.7M	93.57
ResNet18 + ReLU	18.16	11M	93.02
ResNet50 + ReLU	50	25.6M	93.62
ResNet100 + ReLU	100	44.5M	93.75
ResNet18b + FALU	**18**	**4.29M**	**94.40**

Table 4: Experimental results comparing ResNet-based models with FALU, and with reported ResNet models performance for the CIFAR-10 dataset.

Fashion-MNIST

The Fashion-MNIST dataset contains 60,000 (50k/10k train/test split) 28×28 resolution gray scale images in 10 classes of clothing, with 6,000 images per class (Han Xiao and Vollgraf 2017). In Table 5, we report results for three different CNN architectures, consisting of a highly compact CNN model model (29K parameters), and large-scale models, including ResNet and VGG (Karen Simonyan 2015). We applied the data augmentation procedure developed in (Harris et al. 2021) for each experiment. Across all three architectures, FALU improved classification accuracy by roughly 1% over the identical model using ReLU. In particular, for the ResNet18 + FALU topology, we generated accuracy matching the SOTA results for Fashion-MNIST as reported in (Harris et al. 2021).

ImageNet

ImageNet is a popular benchmarking classification database consisting of 14,197,122 RGB images over 21,841 subcategories. We report results in Table 6 demonstrating improvements using the ResNet50 architecture with our proposed FALU AF compared to the baseline ReLU. The model was trained for 120 epochs with an initial learning rate of 0.01 decayed by an order of magnitude every 30 epochs, batch size of 128, and random weight initialization.

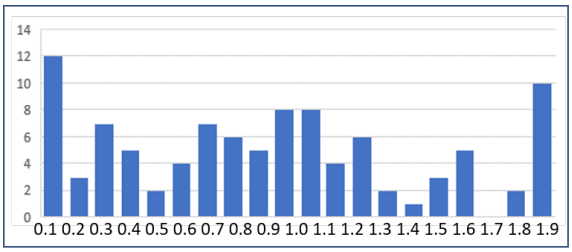

Figure 4: Histogram of resulting α values in the range of $(0, 2)$ from our compact MNIST model (layer 3 filters). Using FALU activation functions, the model converged to a diverse range of AF forms encompassing ReLU, sigmoid, and Gaussian AFs – corresponding to the dominant modes of the distribution.

Neural Network	#Param	Top1%
6c,10c,10c,10fc + ReLU	29K	90.99
6c,10c,10c,10fc + FALU	**29K + 26**	**91.72**
ResNet18 + ReLU	11.16M	95.37
ResNet18 + Swish	11.16M	96.00
ResNet18 + FALU	**11.17M**	**96.28**
VGG + ReLU	39.7M	91.14
VGG + FALU	**39.7M**	**92.09**

Table 5: Comparison of classification accuracy using FALU across a simple, compact CNN, ResNet and VGG for the Fashion-MNIST dataset.

Neural Network	#Param	Top1%	Top5%
ResNet18 + ReLU	44M	70.7	89.9
ResNet50 + ReLU	87M	75.8	92.9
ResNet101 + ReLU	160M	77.1	93.7
ResNet50 + FALU	**87M**	**76.7**	**93.28**

Table 6: Comparison of classification accuracy for ResNet50 with FALU activation function compared with common ResNet model performance using ReLU on ImageNet.

Conclusions

In this work we presented a novel generalization of adaptive activation functions which we call Fractional Adaptive Linear Units. Utilizing concepts from fractional calculus and building upon previous successful activation function research, our method defines a family of diverse morphologies encompassing many traditional and state-of-the-art AFs, thus offering increased flexibility over existing methods. Importantly, FALUs achieve this multiplicity of forms through the introduction of a small number of additional tunable parameters, including the fractional derivative of the AF. For this reason, FALUs are simple to implement using standard Deep Learning libraries. We showed that FALUs consistently outperform baseline AFs across a variety of datasets and model architectures, including both highly compact models and large-scale DNNs.

References

Agostinelli, F.; Hoffman, M.; Sadowski, P.; and Baldi, P. 2014. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*.

Baleanu, D.; and Agarwal, R. 2021. Fractional calculus in the sky. *Advances in Difference Equations*, 2021.

Chen, L.-C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; and Yuille, A. L. 2016. DeepLab: Semantic Image Segmenta-

tion with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *CoRR*, abs/1606.00915.

Clevert, D.-A.; Unterthiner, T.; and Hochreiter, S. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *arXiv: Learning*.

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

Davis, P. J. 2016. "Leonhard Euler's Integral: A Historical Profile of the Gamma Function". *American Mathematical/ doi:10.2307/2309786*, Monthly. 66 (10): 849–869.

Deng, J.; Dong, W.; Socher, R.; jia Li, L.; Li, K.; and Feifei, L. 2009. Imagenet: A large-scale hierarchical image database. In *In CVPR*.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *CoRR*, abs/2010.11929.

Dubey, S. R.; Singh, S. K.; and Chaudhuri, B. B. 2021. A Comprehensive Survey and Performance Analysis of Activation Functions in Deep Learning. *ArXiv*, abs/2109.14545.

Han Xiao, K. R.; and Vollgraf, R. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/*.

Harris, E.; Marcu, A.; Painter, M.; Niranjan, M.; Prügel-Bennett, A.; and Hare, J. 2021. FMix: Enhancing Mixed Sample Data Augmentation. arXiv:2002.12047.

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

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

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016b. Identity Mappings in Deep Residual Networks. *CoRR*, abs/1603.05027.

Hendrycks, D.; and Gimpel, K. 2020. Gaussian Error Linear Units (GELUs). arXiv:1606.08415.

Herrmann, R. 2011. *Fractional Calculus*. World Scientific Publishing.

Hinton, G. E.; Osindero, S.; and Teh, Y. W. 2006. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18: 1527–1554.

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Comput.*, 9(8): 1735–1780.

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

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Ioffe, S.; and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Bach, F.; and Blei, D., eds., *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, 448–456. Lille, France: PMLR.

Iomin, A. 2018. Fractional evolution in quantum mechanics. *Chaos, Solitons and Fractals: X*, 1: 100001.

Kaiming He, S. R., Xiangyu Zhang; and Sun, J. 2015. Deep Residual Learning for Image Recognition. *arXiv:1512.03385*.

Karen Simonyan, A. Z. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR2015*.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17.

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

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report, Master's thesis, Dept. of Comp. Sci., University of Toronto.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012a. ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F.; Burges, C. J. C.; Bottou, L.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 25*, 1097–1105. Curran Associates, Inc.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012b. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, 1097–1105. USA: Curran Associates Inc.

Kurt; and Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2): 251–257.

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

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep Learning. *Nature*, 521(7553): 436–444.

Lecun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 2278–2324.

LeCun, Y.; and Cortes, C. 2010. MNIST handwritten digit database. *http://yann.lecun.com/exdb/mnist/*.

Leshno, M.; Lin, V. Y.; Pinkus, A.; and Schocken, S. 1993. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6): 861–867.

Lu, L.; Shin, Y.; Su, Y.; and Karniadakis, G. 2019. Dying ReLU and Initialization: Theory and Numerical Examples. *ArXiv*, abs/1903.06733.

Luchko, Y. 2020. Fractional derivatives and the fundamental theorem of fractional calculus:. *Fractional Calculus and Applied Analysis*, 23(4): 939–966.

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

Maas, A. L.; Hannun, A. Y.; and Ng, A. Y. 2013. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

Mcculloch, W.; and Pitts, W. 1943. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics*, 6: 115–133.

Misra, D. 2019. Mish: A Self Regularized Non-Monotonic Neural Activation Function. *ArXiv*, abs/1908.08681.

Nair, V.; and Hinton, G. E. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In Fürnkranz, J.; and Joachims, T., eds., *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 807–814.

Nwankpa, C.; Ijomah, W.; Gachagan, A.; and Marshall, S. 2018. Activation Functions: Comparison of trends in Practice and Research for Deep Learning. *ArXiv*, abs/1811.03378.

Ortigueira, M. D. 2011. *Fractional Calculus for Scientists and Engineers*. Springer Publishing Company, Incorporated, 1st edition. ISBN 9400707460.

Pascanu, R.; Mikolov, T.; and Bengio, Y. 2013. On the Difficulty of Training Recurrent Neural Networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, III–1310–III–1318. JMLR.org.

Qiuyu Zhu, R. Z. 2018. HENet: A Highly Efficient Convolutional Neural Networks Optimized for Accuracy, Speed and Storage. *arXiv:1803.02742*.

Ramachandran, P.; Zoph, B.; and Le, Q. V. 2018. Searching for Activation Functions. *ArXiv*, abs/1710.05941.

Rosenblatt, F. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6): 386–408.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4510–4520.

Scardapane, S.; Van Vaerenbergh, S.; Totaro, S.; and Uncini, A. 2019. Kafnets: Kernel-based non-parametric activation functions for neural networks. *Neural Networks*, 110: 19–32.

Springenberg, J. T.; Dosovitskiy, A.; Brox, T.; and Riedmiller, M. A. 2014. Striving for Simplicity: The All Convolutional Net. *CoRR*, abs/1412.6806.

Szegedy, C.; Wei Liu; Yangqing Jia; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, 5998–6008.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2017. Graph Attention Networks. *6th International Conference on Learning Representations*.

Wang, Y.; Li, Y.; Song, Y.; and Rong, X. 2020. The Influence of the Activation Function in a Convolution Neural Network Model of Facial Expression Recognition. *Applied Sciences*, 10: 1897.

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

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *ArXiv*, abs/1708.07747.

Zhang, X.; Zhou, X.; Lin, M.; and Sun, J. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 6848–6856.
instead or the References section will not appear in your paper