Evaluating the Rationale for Accuracy Improvements by the Swish Activation Family over the ReLU Activation Family

Marc Melikyan

November 2022

Abstract

In this publication, we explore the reasons for the significant improvements in accuracy provided by recently proposed non-monotonic activation functions of the Swish family (e.g. GeLU, Swish, Mish, TanhExp, etc.) and contrasting them to frequently utilized and established monotonic functions of the ReLU family (e.g. ReLU, SeLU, ELU, etc.). We address both the effects of the functions' properties as well as the notability of their first derivatives. We do so by performing and making use of empirical experimentation and the analytical calculation of such quantities.

1 Introduction

The activation function is the principal component of any neural network architecture. It transforms the model's computation to a non-linear space, which would otherwise equate to a large number of linear, and thus ineffective, functional compositions. The usage of non-linearities in conjunction with the composite nature of the neural network are conducive to learning highly complicated relationships between the inputs and outputs, thus allowing the neural network to become a highly effective function approximator. In particular, if the output of a linear layer of a neural network is computed as $z = wx + b$, the corresponding transformed output is computed as $a = f(z)$, in which f is the activation function used to introduce non-linearities into the model. The selection of f is the guiding principle behind the development of activations, and its selection changed as novel methods were discovered. f being set to sigmoidal functions such has $\sigma(x)$ and $\text{Tanh}(x)$ was at one point common, evidenced by their usage in the LeNet architecture [1]. However, both functions were ultimately abandoned due to their tendency to saturate outside of their linear regimes [2]. Sigmoidal functions were ultimately switched out for another monotonic family of functions, the ReLU family, being initially popularized by ReLU's usage in the AlexNet architecture [3]. Even more recently, a non-monotonic family

of functions, originating from the GELU activation function [4], have provided notable accuracy gains over ReLU, and have gained popularity in a number of recently proposed neural network architectures [5]. In the following sections, we elaborate on the reasons for the superiority of the Swish family of functions by discussing smoothness, self-gatedness, saturation, derivatives and preconditioning, and empirical results of functions in both families.

2 Smoothness

A function f is defined as smooth if it is fully diffrentiable and continuous for all points $x \in \{-\infty, \infty\}$. Smooth functions are conducive to more efficient learning because neural optimization problems involving compositions of such functions end up having a corresponding smoother loss landscape. This follows from the fact that a composition of two or more non-smooth functions is not smooth whereas a composition of two or more smooth functions is a smooth function. The popularly acclaimed functions of the ReLU family, interestingly enough, do not maintain a smooth profile.

This is can be explained by the sharp change exhibited between the two functional pieces. For example, given the definition of the ReLU activation function:

$$\text{ReLU}(x) = max(0, x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{1}$$

The two distinct linear pieces cause a non-differentiable corner at $x = 0$. The same problem occurs in the cases of leaky ReLU[6], ELU[7], and SELU[8]. This is in contrast to functions in the Swish family, which are all fully differentiable and continuous, and therefore smooth. The definition of the Swish function:

$$\text{Swish}(x) = \sigma(x) * x = \frac{x}{1 + e^{-x}} \tag{2}$$

yields such a smooth function. This can also be seen from the graphs of the two functions shown below:

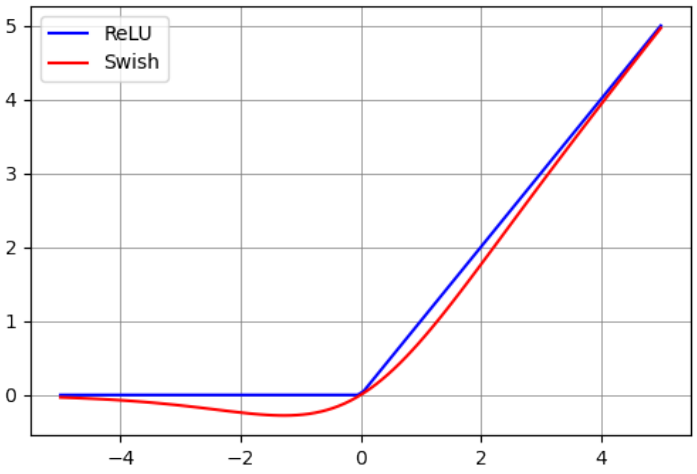

Figure 1: Graphs of the Swish and ReLU functions.

The figure emphasizes this smoothness more evidently. This smooth profile can also be extracted from the output maps of a 6-layered neural network taking the coordinates as input:

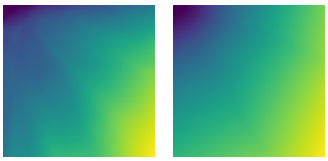

Figure 2: The activation map intensities of ReLU (left) and Swish (right) on a 6-layered randomly initialized neural network.

The transitions in ReLU are sharp and jagged, whereas Swish's profile is perfectly smooth.

This smoothness can be seen directly in the resulting loss functions. Simple data of $N = 5$ with the relationship $y = 2X$ was utilized with a single feature. Given such a regression setting with either of the activation functions in use, the following loss graphs are yielded:

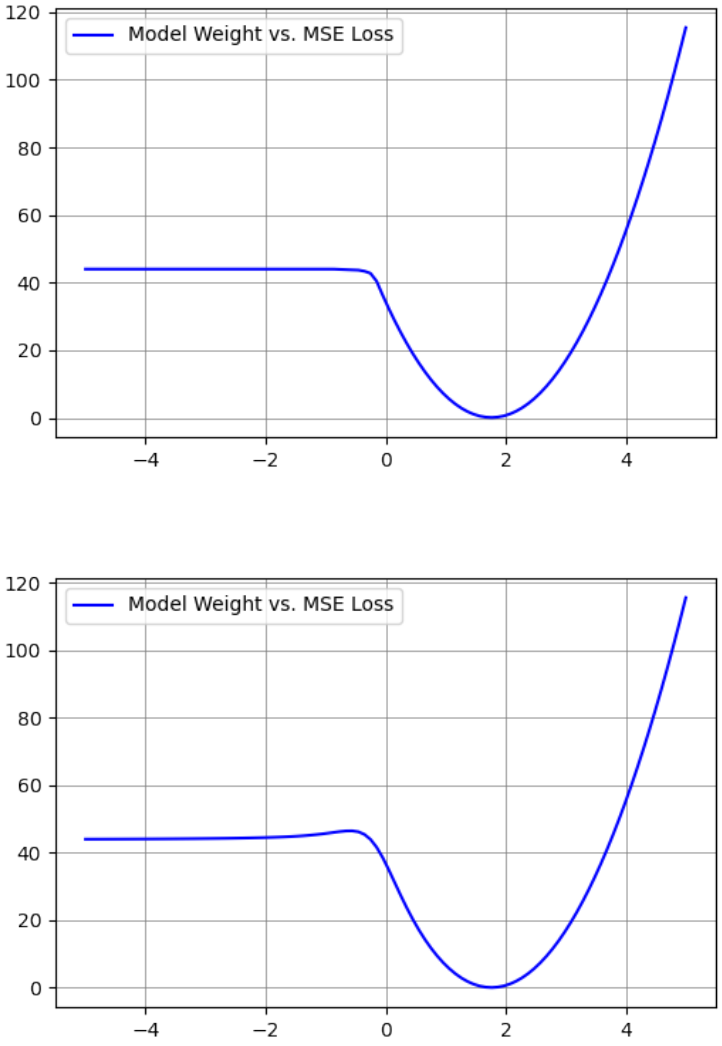

Figure 3: The loss functions of ReLU (top) and Swish (bottom) in a univariate regression setting.

Interestingly enough, both activation functions produce a similar looking loss for one layered cases, but one is smooth, and the other is not, mirroring the trend found in the functions themselves. Further, the following data illustrates the number of times the corresponding network converged (calculated by noting whether the rounded predictions were equivalent to the actual outputs) averaged over 100 trials with varying layer numbers on the same data as the loss graphs

above used:

Table 1: The percentage of times that global optima were attained by Swish and ReLU.

Number of Layers	Swish	ReLU
1	60.39%±5.00%	41.02%±4.65%
2	34.82%±5.04%	20.06%±3.99%
3	16.92%±3.74%	8.76%±2.46%

In all trials, Swish demonstrated a higher rate of convergence.

3 Self-Gatedness

Despite similar graphical properties, the Swish and ReLU functions achieve these properties in differing ways, particularly because the Swish family are *self-gated*. That is to say that the function is the result of a product of the original input and the imposition of an explicitly non-linear function upon it. This is similar in nature to the gating done in LSTM networks, except for the fact that the input is gated reflexively [9][10][11][12]. Having the activation function implicitly gate the input is useful because of the effectiveness of gating; in the LSTM and GRU architectures, using gating allows for the updation of the memory cell, in which unnecessary information may be easily filtered out, whereas necessary information is propagated [13]. This is similar in the case of activations, except because the input is able to reflexively modulate itself, it can remove and filter out unnecessary information without requiring two inputs [9].

4 Saturation

Figure 2 illustrates the differences between the two loss functions graphically. Analyzing the negative section, we notice that both functions saturate as x goes to ∞. In ReLU, this functional property leads to the dying ReLU phenomenon, which results in gridlocks during gradient-based optimization because the gradients go to 0 [14]. This is in contrast to Swish and its respective family of functions, which have non-zero lower bounds, thus allowing negative gradient information to flow during backpropagation [10][12]. The lower bounds of several Swish family functions as well as the ReLU function are listed along with their respective derivative lower bounds:

Table 2: The lower bounds of Swish and ReLU family functions and their derivatives.

Function	f	f'
ReLU	0	0
GELU	-0.17	-0.129
Swish	-0.278	-0.1
Mish	-0.309	-0.113
Serf	-0.348	-0.127
TanhExp	-0.353	-0.132

Within the ReLU family, while it is true that functions such as leaky ReLU or ELU have been developed to ameliorate the dying ReLU problem, both of those functions were themselves specifically designed to resolve it [6][7]. This is in contrast to the Swish family, whose functions innately combat against the problem of gradient saturation due to the ability of negative gradient information to flow.

5 Derivatives and Preconditioning

The first derivative of the Swish function is as follows:

$$\text{Swish}'(x) = [\frac{x}{1+e^{-x}}]' = \sigma(x)+x\sigma'(x) = \sigma(x)+x\sigma(x)(1-\sigma(x)) = \sigma(x)+x\sigma(x)\gamma(x)$$

(3)

Wherein $\gamma(x)$ is a gradient preconditioner function and $\sigma(x)$ is the sigmoid function. A preconditioner function is a function which allows for efficient numerical optimization by allowing gradient-based updation to relay more useful information during training. Newton and Quasi-Netwon optimization algorithms, based on applying Newton's root-finding algorithm to the objective function's derivative, have seen significant success, such as BFGS or the recently proposed AdaHessian method, both of which approximate the second-order Hessian of the objective function, H, to avoid large computational overhead [15][16][17][18][19]. The Hessian matrix is able to provide useful second order information on curvature and concavity, allowing for more efficient optimization.

$\gamma(x)$ can be interpreted as acting as a preconditioner because it relays useful gradient information to regularize and smooth out the gradients, as was made evident by Figure 2. A similar trend is seen in other functions of the Swish

family, namely, Mish, whose first derivative is as follows:

$$\text{Mish}'(x) = [\text{Tanh}(\ln(1+e^{-x}))]' = \text{Sech}^2(\ln(1+e^{-x}))x\sigma(x) + \frac{\text{Mish}(x)}{x} = \gamma(x)\text{Swish}(x) + \frac{\text{Mish}(x)}{x} \quad (4)$$

In which $\gamma(x)$ again acts as a preconditioner, and therefore smoothing the loss objective and yielding simpler optimization. The authors of the Mish activation posit that this may be relevant to Mish's empirical performance gains over Swish [10]. Serf also exhibits a first derivative in which a preconditioner is present, the only difference between its preconditioner and that of Mish is that it is derived from erf(x) as opposed to Tanh(x) in Mish.

6 Empirical Results

In this section, we discuss empirical results of these functions employed in artificial neural networks (ANNs) on the numerical MNIST dataset [20].

6.1 Artificial Neural Networks

Training ANNs was done by building an L-layered neural network with the corresponding activation was employed on the layer's output, and with dropout and batch normalization employed as well. The models were trained for 15 epochs with a learning rate of 0.01, a batch size of 128, and with the SGD optimizer. The only variable portions of training were the activation function used and L.

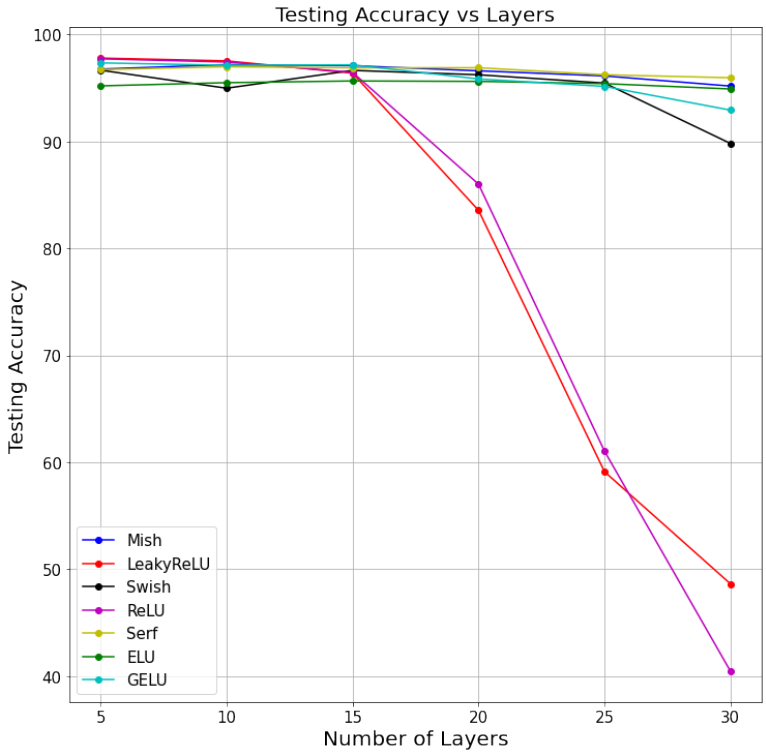

Figure 4: Activation Function Accuracy vs Layer Size on MNIST

The results clearly indicate superior testing accuracy by the Swish family.

7 Conclusion

We extensively explored the evolution of the activation function in neural networks. Sigmoidal functions, which were embraced initially, were ultimately discarded as a result of unreliability and gradient saturation. In their place, came the ReLU family which trumped the sigmoidal functions in accuracy, gradient flow, and improving issues of gradient saturation. In the same vein, however, functions in the ReLU family are just as susceptible to replacement when accounting the unequivocal improvements made upon them by the Swish family. That is, improvements in problems of saturation, ameliorating the dying ReLU problem, achieving smooth, fully differentiable activations, acting as highly efficient self-gaters, and smoothing optimization through various preconditioners.

References

[1] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[2] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network, 2015.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[4] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[5] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. 2019.

[6] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, Georgia, USA, 2013.

[7] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015.

[8] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. 2017.

[9] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.

[10] Diganta Misra. Mish: A self regularized non-monotonic activation function, 2019.

[11] Xinyu Liu and Xiaoguang Di. Tanhexp: A smooth activation function with high convergence speed for lightweight neural networks, 2020.

[12] Sayan Nag and Mayukh Bhattacharyya. Serf: Towards better training of deep neural networks using log-softplus error activation function, 2021.

[13] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[14] Lu Lu. Dying ReLU and initialization: Theory and numerical examples. *Communications in Computational Physics*, 28(5):1671–1706, jun 2020.

[15] Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael W. Mahoney. Adahessian: An adaptive second order optimizer for machine learning. 2020.

[16] Charles George Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[17] Roger Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970.

[18] D Goldfarb. A family of variable metric updates derived by variational means. mathematics of computing. 1970.

[19] David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

[20] Yann LeCun and Corinna Cortes.